CEPESC

Centro de Pesquisa e Desenvolvimento para a Segurança das Comunicações



ABIN

Sumário

Sumário **2**

O Centro de Pesquisa e Desenvolvimento para a Segurança das Comunicações **4**

Artigos publicados em 2020 **5**
SBSeg'20 - LGPD Levantamento de Técnicas Criptográficas e de Anonimização para Proteção de Bases de Dados . . . 6
SBSeg'20 - Improving the Security of ChaCha against Differential-Linear Cryptanalysis 20

Artigos publicados em 2021 **34**
EUROCRYPT'21 - Improved Linear Approximations to ARX Ciphers and Attacks Against ChaCha . . . 35

Artigos publicados em 2022 **65**
SBSeg'22 - libharpia: a New Cryptographic Library for Brazilian Elections . . . 66

Artigos publicados em 2023 **80**
SBSeg'23 - Automated security proof of SQUARE, LED and CLEFIA using the MILP technique 81

Artigos publicados em 2024 **92**
SBSeg'24 - Cutting dimensions in the LLL attack for the ETRU post-quantum cryptosystem . 93
SBSeg'24 - Lattice Basis Reduction Attack on Matrix NTRU . . . 104
SBSeg'24 - Modified versions of ML-KEM based on Brazilian cryptographic resources . . . 118
SBSeg'24 - PARDED: Uma ferramenta de detecção passiva de malwares com foco em rootkits que utilizam técnicas de ofuscação de tráfego . . . 134

Artigos publicados em 2025 **142**
SBSeg'25 - A Quantum-Resistant Advanced Metering Infrastructure . . . 143
SBSeg'25 - Aplicacao pratica da quebra do KeeLoq uma abordagem de baixo custo . . . 160
SBSeg'25 - FrodoKEM based on multicore implementation enhancing performance and memory efficiency in IoT applications . . . 177
SBSeg'25 - Improved Decryption Bounds and Key Generation for Matrix NTRU over Integral Domain . . . 185
SBSeg'25 - Multicore Implementation of ML-KEM on Embedded Devices . . . 199
SBSeg'25 - Tailoring a Genetic Algorithm for searching Shortest Lattice Vector in NTRU Lattices 216
SIBGRAPI'25 - Avaliacao de OCR Embarcado para Dispositivos Moveis Desempenho, Privacidade e Usabilidade . . . 224
Journal of Network and Systems Management - Performance Evaluation of Cloud Native Applications A Systematic Mapping Study . . . 230

Artigos publicados em 2026 **265**
CiC - Matrix Polynomial Attack on the Megrelishvili Key Exchange Protocol . . . 266
Journal of the Brazilian Computer Society - Lattice Basis Reduction Attack on Matrix NTRU 275
SBSeg'26 - Optimizing ML-KEM with OpenMP A Multilevel Parallelization Study . . . 286
SBSeg'26 - APKHash Grafos de Similaridade Estrutural como Evidência de Ruído na Rotulação de Famílias de Malware . . . 302
SBSeg'26 - msgX Rumo a Mensagens Pós-Quânticas Soberanas para Implantações Governamentais do Protocolo Matrix . . . 318

SBSeg'26 - Ecossistema de Comunicações Soberano Brasileiro 334
Journal of Cloud Computing - Matrix protocol a comprehensive systematic mapping study . . 342
SBRC'26 - A Comparative Performance Study of the Matrix sync Endpoint on Synapse and Tuwunel 371
SBRC'26 - Performance Analysis and Refactoring of the Synapse Reference Server through Observability 385

O Centro de Pesquisa e Desenvolvimento para a Segurança das Comunicações

A segurança da informação desempenha um papel essencial em uma sociedade cada vez mais digitalizada e interconectada, influenciando diretamente questões como privacidade, confiabilidade e integridade de dados sensíveis. Nesse contexto, a criptografia e a segurança cibernética se destacam como uma das principais ferramentas para garantir a integridade das comunicações, a segurança de dispositivos e sistemas críticos, e a preservação da confiança nas tecnologias emergentes. Compreendendo essa responsabilidade, o Centro de Pesquisa e Desenvolvimento para a Segurança das Comunicações (CEPESC) tem se dedicado continuamente ao avanço científico e tecnológico nessas áreas, contribuindo tanto para o desenvolvimento de novas soluções como para o fortalecimento da capacidade técnica do país nesse campo estratégico.

Esta coletânea reúne artigos científicos produzidos ao longo dos anos por nossa equipe de pesquisadores, abrangendo desde abordagens teóricas inovadoras até aplicações práticas de criptografia voltadas à resolução de problemas reais. Os trabalhos aqui compilados representam um diálogo contínuo entre a pesquisa acadêmica e as demandas concretas da sociedade, com o objetivo de oferecer contribuições relevantes para a comunidade científica, instituições governamentais e a sociedade civil. Ao disponibilizarmos essas publicações, esperamos também incentivar novas pesquisas, parcerias estratégicas e o fortalecimento da cultura de segurança da informação no Brasil e no mundo.

Artigos publicados em 2020

LGPD: Levantamento de Técnicas Criptográficas e de Anonimização para Proteção de Bases de Dados

Thiago R. Sousa[1] **Murilo Coutinho**[1] **Lilian Coutinho**[2] **Robson Albuquerque**[3]

[1]Centro de Pesquisa e Desenvolvimento para a Segurança da Comunicações (CEPESC)

[2]Escola de Inteligência (Esint), Agência Brasileira de Inteligência, Brasília, Brasil

[3]Programa de Pós-Graduação Profissional em Engenharia Elétrica (PPEE), Departamento de Engenharia Elétrica - Universidade de Brasília, Brasília, Brasil

Resumo. *A Lei nº 13.709, de 14 de agosto de 2018, conhecida como Lei Geral de Proteção de Dados Pessoais (LGPD), veio para instituir princípios e regras para a proteção das pessoas naturais no que diz respeito ao tratamento de seus dados, principalmente no formato digital. Por essa razão, surge a necessidade de se estabelecer soluções tecnológicas capazes de atender às imposições da lei. Neste trabalho, apresentamos um levantamento de técnicas e ferramentas de anonimização e de criptografia que demonstram potencial para auxiliar no cumprimento da LGPD, no caso específico da proteção de bases de dados. Dentre as técnicas comparadas, percebe-se que não há nenhuma que atenda perfeitamente a todas as situações, seja por questões de desempenho ou por considerações de segurança. Ainda assim, conclui-se que, quando possível, essas soluções devem ser utilizadas, pois têm o potencial de aumentar significativamente a segurança dos sistemas e auxiliar no cumprimento da lei.*

1. Introdução

O uso da criptografia sempre foi importante para garantir a segurança das informações e das comunicações, proteger o sigilo de conhecimentos e de documentos sensíveis e, na era da informação, tornou-se fundamental para a preservação da privacidade de indivíduos no mundo digital. Ainda assim, tal importância tende a aumentar em virtude das novas leis para a proteção de dados que surgiram, primeiro na Europa com a *General Data Protection Regulation* (GDPR) e, posteriormente, em vários países ao redor do mundo.

No Brasil, em 14 de agosto de 2018, foi sancionada a Lei nº 13.709, conhecida como Lei Geral de Proteção de Dados Pessoais (LGPD), com o objetivo de proteger o livre desenvolvimento da personalidade da pessoa natural e os direitos fundamentais de liberdade (CF, art. 5º, inciso IV) e de privacidade (CF, art. 5º, incisos X, XI e XII)[1]. Conforme o art. 1º da LGPD, a Lei dispõe sobre o tratamento de dados pessoais[2], inclusive nos meios digitais, por pessoa natural ou por pessoa jurídica de direito público ou privado. Nesse sentido, destina-se também aos órgãos e entidades da Administração Pública e, por tratar-se de lei nacional, deve ser observada por todos os entes federados – União, Estados, Distrito Federal e Municípios.

[1]Está em trâmite a PEC nº 17/2019, que visa incluir a proteção de dados pessoais entre os direitos e garantias fundamentais. Além disso, no julgamento das ADIs 6387, 6388, 6389 e 6390, o STF a reconheceu como um direito fundamental autônomo.

[2]Conforme o art. 5º, inciso X da LGPD, "tratamento"deve ser compreendido como toda operação realizada com dados pessoais, como coleta, acesso, armazenamento etc.

Mesmo não estando em vigor, muitos órgãos e empresas passaram a buscar novos métodos e tecnologias visando a adequação do processamento de dados pessoais em conformidade à lei. Tais soluções tecnológicas têm largo escopo dentro da área de segurança da informação e das comunicações, incluindo-se aspectos da segurança cibernética. Naturalmente, a criptografia se destaca nesse cenário já que consiste em técnicas matemáticas que, se implementadas corretamente, garantem a impossibilidade de acesso não autorizado, mesmo por adversários com grande poder de processamento. Destaca-se, contudo, que segurança não é a mesma coisa que privacidade, e que apesar de ser difícil garantir a privacidade em sistemas inseguros, é possível não ter privacidade em sistemas com boas práticas de segurança.

Historicamente, a criptografia se concentrou em resolver o problema de comunicação clássico em que dois entes (Alice e Bob) buscam se comunicar de forma segura mesmo na presença de um adversário (Eve) capaz de acessar, ler, ou mesmo modificar o canal de comunicação. Hoje, pode-se afirmar que a criptografia é plenamente capaz de resolver esse problema, através de algoritmos de sigilo, troca de chaves, resumo e assinatura digital, que garantem a autenticidade, confidencialidade e integridade da comunicação. Portanto, soluções de comunicação por texto ou voz, videoconferências, acesso remoto via redes privadas virtuais (VPN) e similares podem, se bem implementadas, prover segurança suficiente para proteger tanto a privacidade dos usuários quanto a própria empresa que se responsabiliza pelos dados, garantindo a conformidade com a lei.

Há, entretanto, outras situações que não se encaixam no paradigma clássico da criptografia. Por exemplo, suponha a existência de uma base contendo dados privados que esteja armazenada em um servidor e uma série de analistas que necessitam acesso aos dados para realizarem trabalhos estatísticos. Em algumas situações, cada analista pode ser considerado como um potencial adversário, já que nem sempre podemos ter certeza que não irão acessar informações privadas que não deveriam ou mesmo vazá-las. Há também casos em que o próprio servidor pode ser visto como adversário, como quando dados privados são armazenados na nuvem com o intuito de utilizar o poder computacional dos servidores de grandes empresas.

Para superar esses desafios, uma série de técnicas e ferramentas têm sido desenvolvidas nos últimos anos. Porém, é justo afirmar que ainda não há nada estabelecido no mercado ou na academia como um padrão a ser seguido ou uma solução que resolva todos esses problemas de forma plenamente segura e eficiente. Sendo assim, este trabalho apresenta um levantamento de um subconjunto de tais técnicas, dentre elas Anonimização, Pseudonimização, Privacidade Diferencial, *Fully Homomorphic Encryption, Property Preserving Encryption* e *Oblivious Random Access Memory*. Adicionalmente, elencamos algumas ferramentas de busca e gerenciamento de dados cifrados baseadas nessas técnicas, discutindo sua utilização, segurança e ataques.

Neste trabalho, também apresentamos uma discussão sobre situações em que cada uma dessas técnicas podem ser utilizadas, muitas vezes em conjunto, para incrementar a segurança dos sistemas para a proteção das informações armazenadas em bases de dados e também em situações de proteção de privacidade ou anonimato quando bases de dados devem ser disponibilizadas para analistas externos. Conclui-se que, apesar da potencial existência de ataques, a segurança agregada ainda é significativa, o que dificulta a recuperação de informações, bloqueando a maior parte dos adversários e contribuindo na

garantia da manutenção da privacidade e do cumprimento da lei.

O restante do artigo é organizado da seguinte forma: na Seção 2, discutimos a relação entre a LGPD e a criptografia. Na Seção 3, descrevemos as principais técnicas de anonimização e criptografia que podem ser utilizadas para a proteção de bases de dados. Na Seção 4, apresentamos exemplos de ferramentas utilizadas nesse contexto e, na Seção 5, abordamos ataques que demonstram as limitações das técnicas e soluções atuais. Na Seção 6, discutimos algumas situações em que as técnicas apresentadas podem ser utilizadas, fazendo um comparativo entre elas e na seção 7 concluímos o trabalho.

2. LGPD e Criptografia

O modelo para a criação da lei de proteção de dados pessoais brasileira foi a *General Data Protection Regulation* (GDPR). Tal regulamentação tem aplicação nos países que integram a União Europeia (UE), impondo obrigações a quaisquer organizações que utilizem e coletem dados pessoais nos Estados membros.

O art. 5(1) da GDPR delimita os princípios relacionados ao processamento de dados pessoais, dentre os quais destaca-se o princípio da segurança, que determina o uso de técnicas apropriadas e de medidas organizacionais que garantam um nível de segurança apropriado aos riscos envolvidos, incluindo a proteção contra processamento ilegais ou não autorizados e contra perda, destruição ou danos. Para tanto, a criptografia dos dados pessoais é prevista em um rol exemplificativo de medidas que podem ser utilizadas de acordo com a natureza, o escopo, o contexto e as finalidades do processamento, bem como com os riscos aos direitos e liberdades individuais (GDPR, art. 32).

Nesse contexto, cada controlador ou processador deve avaliar os riscos que estão presentes no processamento de dados pessoais, como vazamentos, perda, destruição, alteração, acesso ou divulgação não autorizados, e outros eventos potencialmente causadores de danos físicos, materiais ou imateriais, devendo mitigar tais riscos e garantir um nível apropriado de segurança a partir da implementação de medidas, a exemplo da criptografia.

Assim como a GDPR, a LGPD inclui a segurança como um de seus dez princípios gerais[3] (art. 6º, VII), determinando que os agentes de tratamento devem adotar medidas técnicas, administrativas e de segurança aptas a proteger os dados pessoais de acessos não autorizados e de situações acidentais ou ilícitas, como destruição, perda, alteração, comunicação, difusão ou qualquer outra forma de tratamento inadequado ou ilícito (art. 46). Se, por um lado, a lei brasileira não dispõe expressamente sobre o uso da criptografia como medida técnica sugerida, por outro é clara no sentido de que, em caso de incidente de segurança que possa acarretar risco ou dano relevante aos titulares, será avaliada eventual comprovação de que foram adotadas medidas técnicas adequadas para tornar ininteligíveis os dados pessoais afetados (art. 48, § 3º). Com efeito, as técnicas de criptografia permitem exatamente isso: tornar ininteligível o dado àqueles que não possuem as chaves criptográficas, ou seja, a qualquer pessoa que não está autorizada a acessá-lo.

Os controladores e processadores devem avaliar os riscos envolvidos nos tratamentos de dados pessoais que realizam, implementando medidas adequadas de proteção.

[3]O art. 6 da LGPD determina que as atividades de tratamento de dados pessoais deverão observar a boa-fé e os seguintes princípios: 1) finalidade; 2) adequação; 3) necessidade; 4) livre acesso; 5) qualidade dos dados; 6) transparência; 7) segurança; 8) prevenção; 9) não discriminação; e 10) responsabilização e prestação de contas.

Não foram poucos os incidentes de segurança ocorridos nas últimas décadas, e os danos gerados poderiam ter sido reduzidos ou mesmo evitados se os dados fossem criptografados. Cientes de suas responsabilidades, muitas empresas e órgãos públicos já estudam o uso de técnicas criptográficas para base de dados que atendam aos objetivos da LGPD já que a criptografia se apresenta, simultaneamente, como meio para garantir a conformidade à lei, para assegurar a proteção de direitos fundamentais e para a segurança dos sistemas.

3. Técnicas para a segurança de base de dados

3.1. Pseudonimização e Anonimização

Uma das formas mais simples para tentar proteger a privacidade de indivíduos em bases de dados é pela anonimização [Jain et al. 2016] ou pela pseudonimização [Stalla-Bourdillon and Knight 2016] que, em geral, não são consideradas técnicas criptográficas. Tais técnicas podem ser entendidas como a modificação ou destruição de informações em bases de dados de forma que não seja mais possível identificar os indivíduos. Isso é feito, em geral, de três maneiras: supressão, quando alguns tipos de dados sensíveis são eliminados da base de dados; substituição, quando dados sensíveis são substituídos por outros dados não sensíveis ou falsos; e generalização, quando dados específicos são substituídos por categorias mais genéricas, por exemplo, a idade de indivíduos podem ser substituídas por intervalos como "entre 20 e 30 anos".

A pseudonimização se difere da anonimização no sentido de que a pseudonimização permite reidentificar e recuperar os dados originais enquanto a anonimização não (ao menos teoricamente). Uma das técnicas de anonimização mais conhecidas é denominada k-anonimato [Sweeney 2002] e busca garantir que a informação para cada pessoa contida nos dados disponibilizados seja indistinguível de pelo menos $k - 1$ outros indivíduos cujas informações também existam nos dados disponibilizados. Outras técnicas foram desenvolvidas como evoluções do k-anonimato, em particular a l-diversidade [Machanavajjhala et al. 2007] e a t-proximidade [Li et al. 2007].

Existem situações em que essas técnicas simples podem ser utilizadas. Por exemplo, o processo de pseudonimização conhecido como *Tokenização*, que consiste na substituição de dados por informações aleatórias conhecidas como *tokens*, é utilizado com sucesso em algumas aplicações financeiras para proteger dados sensíveis como contas bancárias e números de cartões de crédito [Mattsson and Rozenberg 2013]. Entretanto, em grande parte dos casos, técnicas como o k-anonimato não podem ser consideradas verdadeiramente seguras nem capazes de garantir a privacidade. Isso porque existem ataques efetivos de deanonimização explorando propriedades da própria base de dados ou de outras bases cujos dados estão correlacionadas com as informações sobre os indivíduos que se deseja identificar. Um exemplo famoso neste sentido, ocorreu com dados anonimizados da empresa Netflix, quando um grupo de pesquisadores foi capaz de identificar usuários, desvendando até mesmo suas preferências políticas e outras informações sensíveis [Narayanan and Shmatikov 2008].

3.2. Privacidade Diferencial (PD)

A técnica de Privacidade Diferencial (*Differential Privacy*) pode ser utilizada em situações em que se deseja que um analista possa realizar análise de dados sem afetar a pri-

vacidade dos indivíduos que fazem parte da base de dados [Dwork et al. 2014][4]. A PD funciona a partir da adição de ruído aos dados de maneira que não seja possível ter certeza da informação particular de determinado indivíduo ao mesmo tempo em que se permite boas estimativas para características populacionais. Portanto, tais técnicas possibilitam um balanço entre utilidade e privacidade, sendo que quanto maior o ruído adicionado menor será a utilidade e maior a privacidade, e vice-versa.

As técnicas de PD podem ser úteis em diversos cenários para contribuir com as instituições e empresas a seguir a LGPD. Um possível caso de uso é por órgãos como o Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira (INEP), que possui dados privados de estudantes e instituições de ensino, mas que pode necessitar dar acesso a base de dados a pesquisadores internos ou externos. Neste caso, seria possível disponibilizar dados numéricos de interesse através da adição de ruído de forma a mascarar dados individuais, mas ainda possibilitando a detecção de tendências ou comparações entre instituições de ensino. Outra possibilidade é utilizar a PD para mitigar o risco de coleta de dados de usuários adicionando ruído já na origem da coleta, evitando maiores consequências no caso de comprometimento da base de dados da empresa, mas preservando a possibilidade de melhorar os produtos e serviços a partir da análise de comportamento dos usuários. Por exemplo, a empresa *Apple* já utiliza PD para coletar informações obtidas a partir de sugestões de palavras, sugestões de "emojis", gasto de energia do aparelho, entre outros[5]. No contexto da GPDR, o projeto SODA (`https://www.soda-project.eu`) foi criado com o objetivo de desenvolver um sistema seguro para o processamento de dados pessoais em grande escala em conformidade com a GDPR. Uma de suas vertentes de pesquisa é voltada para a utilização de PD em conjunto com computação multipartidária segura.

Apesar de ter potencial aplicabilidade em diversas áreas, deve-se levar em conta algumas limitações das técnicas de PD. Em primeiro lugar, a PD é uma definição e não um algoritmo. Algoritmos específicos são desenvolvidos para diferentes situações e seu funcionamento deve ser avaliado cuidadosamente em cada uma delas [Dwork and Nissim 2004, Blum et al. 2013]. De fato, a depender do tipo de estatística que se deseja computar, pode nem mesmo existir um método conhecido de se aplicar a PD. Outro problema é que a técnica não funciona bem para bases de dados pequenas já que, neste caso, o ruído necessita ser pequeno para que o modelo possua utilidade, o que reduz significativamente a privacidade. Adicionalmente, a técnica apresenta problemas quando existe alta variabilidade na distribuição dos dados, já que a quantidade de ruído necessária para a privacidade pode destruir completamente a utilidade dos dados.

Finalmente, em muitos casos há a necessidade de limitar o número de consultas feitas por um mesmo analista. Caso contrário, o analista poderia realizar repetidas consultas sobre o mesmo indivíduo, o que possibilita a eliminação do ruído e, como consequência, a exposição do valor que se busca proteger. Ainda, um analista mal intencionado, mesmo que limitado no número de consultas, poderia convencer outros analistas a fazer a mesma consulta ou mesmo criar contas falsas para executar este ataque [Dwork 2008].

[4]Outra técnica que pode ser utilizada com objetivos parecidos é chamada de *Quantitative Information Flow* e carrega muitas similaridades com a PD [Alvim et al. 2011].

[5]`https://www.apple.com/privacy/docs/Differential_Privacy_Overview.pdf`

3.3. Fully Homomorphic Encryption (FHE)

FHE é uma tecnologia promissora do ponto de vista de segurança. Ela permite realizar operações no texto em claro, utilizando-se somente os seus cifrados correspondentes sem a necessidade de decifração. O resultado dessa operação permanece cifrado e pode ser transmitido de forma segura ao usuário que requisitou a operação. A técnica de FHE tem o potencial de garantir a privacidade em diversas situações nas quais isso não era possível como, por exemplo, o processamento de dados privados na nuvem sem que a empresa que disponibiliza o serviço tenha ideia das informações que realmente estão sendo processadas. Dessa forma, elimina-se o risco de comprometimento desses dados pela própria empresa, garantindo-se o cumprimento da lei[6].

FHE vem se tornando viável à medida que os computadores se tornam mais poderosos. Em 2012, um esquema de FHE implementado em [Gentry et al. 2012] gastou 40 minutos para calcular um bloco de AES (o que é seis ordens de magnitude mais lento do que o correspondente cálculo não homomórfico). Não obstante, sistemas homomórficos com funcionalidades restritas têm sido utilizados na prática, ou seja, eles permitem certos tipos de manipulação em dados cifrados, ao invés de funções arbitrárias como em FHE, e têm a vantagem de ter performance elevada, o que os tornam práticos. Por exemplo, o esquema de Paillier [Paillier 1999] permite realizar somas homomórficas de maneira eficiente.

Recentemente, um projeto em parceria com algumas universidades europeias disponibilizou uma biblioteca, denominada TFHE [Chillotti et al. 2016], que permite o cálculo de 10 tipos de portas binárias (AND, OR, XOR, ...) e leva em torno de 13 milissegundos em cada cálculo. Apesar de ser em torno de um bilhão de vezes mais lenta do que o simples cálculo não homomórfico, os algoritmos implementados com esta biblioteca já têm potencial aplicações em problemas reais de física e estatística [Ducas and Micciancio 2015]. Por exemplo, Bourse et. al. [Bourse et al. 2018] utiliza a biblioteca TFHE para treinar redes neurais sobre imagens cifradas homomorficamente obtendo taxa de acerto de 96% na base de dados MNIST em menos de 1.7 segundos. Adicionalmente, outros trabalhos demonstram que é possível aumentar a performance em até 26 vezes utilizando GPUs no processamento [Dai and Sunar 2015].

A conclusão é que na maioria dos casos, utilizar FHE para tratar com bases de dados arbitrárias e de tamanhos moderados ainda é proibitivo do ponto de vista computacional, não sendo adequado para casos em que a velocidade de consulta e resposta seja um requisito. Por isso, os sistemas práticos utilizam técnicas menos seguras [Fuller et al. 2017] em troca de maior performance, sendo úteis para sistemas onde um vazamento específico de informação não irá comprometer a segurança dos dados.

3.4. Property Preserving Encryption (PPE)

Ao invés de focar em realizar operações arbitrárias em dados cifrados, uma alternativa é a utilização de esquemas criptográficos que preservem certa propriedade, conhecidos como PPE. Uma modalidade de PPE se chama OPE (*Order Preserving Encryption*) e serve para realizar comparações de ordem. Este esquema basicamente cifra valores numéricos

[6]Outra técnica promissora é chamada de Functional Encryption [Boneh et al. 2011]. Ela generaliza a noção de cifração de chave pública e permite que o detentor de uma chave de decifração associada a uma função obtenha somente o valor dessa função avaliada sobre o texto em claro e nada mais.

de forma que a ordem dos dados em claro seja preservada em seus cifrados correspondentes. Por exemplo, se objetivamos realizar somente comparações entre valores sem a necessidade de saber quais são estes valores em claro, então um esquema de cifração que preserve a ordem é ideal para essa situação. Neste caso, um sistema de cifração com essa propriedade deve vazar somente a ordem dos dados cifrados [Popa et al. 2011]. Outros sistemas OPE podem vazar outros tipos de informações, além da ordem dos dados.

Outro PPE bastante conhecido é a cifração determinística. As técnicas de cifração modernas garantem que blocos de textos iguais sejam mapeados em cifrados diferentes com alta probabilidade, o que garante a importante propriedade de segurança semântica [Boneh and Shoup 2015]. Em contrapartida, esse tipo de cifração não permite que sejam realizadas buscas que contenham critérios de igualdade na base de dados. Na cifração determinística, blocos de textos iguais devem ser mapeados para os mesmos cifrados, abrindo mão da segurança semântica pela funcionalidade de busca. Além de aplicações em sistemas de bases de dados, a cifração determinística pode também ser aplicada em buscas de palavras-chaves em documentos cifrados.

Por último, temos ainda o *Format Preserving Encryption* (FPE), que basicamente cifra mensagens mantendo a mensagem cifrada no mesmo formato que a mensagem em claro (por exemplo cifrando um número de cartão de crédito válido e obtendo outro número de cartão de crédito válido). A grande vantagem desse esquema é que ele mantém protocolos de comunicação e de acesso intactos, já que os formatos dos dados são preservados. Por exemplo, sistemas que manipulem bases de dados com variáveis em determinados formatos poderiam acessar os dados cifrados com FPE da mesma forma [Brightwell and Smith 1997]. Em tais esquemas, há que se precaver com o tamanho do espaço de mensagens utilizado no esquema de cifração para prevenir ataques (ver p.ex. os ataques sobre o esquema FPE baseado em redes *Feistel* em [Durak 2017]).

Os esquemas PPEs são aplicáveis em situações onde não se faz necessário disponibilizar o dado bruto para que o utilizador destes possa analisá-los, e isso deve ser sempre avaliado pelas instituições. Em várias situações, informações de frequência ou de ordenação dos dados já são suficientes. A desvantagem é que os PPEs vazam informações importantes que podem ser utilizadas por atacantes em determinados cenários, como veremos na sequência. Esquemas PPE são eficientes e têm performance comparada a sistemas que não utilizam cifração. Já os esquemas FHE têm performance ordens de magnitude inferior à esquemas com vazamento controlado como os PPEs [Popa et al. 2011].

3.5. *Oblivious Random Access Memory* (ORAM)

ORAM [Goldreich and Ostrovsky 1996] pode ser utilizada para esconder padrões de acesso em bases de dados disponibilizadas em fontes externas. Esses padrões de acesso vazam informações sobre: qual dado está sendo acessado, quando ele foi acessado pela última vez, se o mesmo dado está sendo acessado e se o acesso é sequencial ou aleatório. Essas informações podem auxiliar o atacante a recuperar dados em claro mesmo que armazenados em bases de dados cifradas [Grubbs et al. 2019]. O princípio básico de ORAM é que, para que esses acessos sejam escondidos, os blocos de dados que são lidos precisam se mover, e não ficarem estáticos, caso contrário haveria vazamento de informações de frequência sobre os dados armazenados. Dessa forma, cada vez que um bloco é lido ele precisa ser realocado em outro local de forma aleatória. ORAM também ofusca

o tipo de operação (leitura ou escrita) que está sendo realizada. Isso é atingido através de uma sequência de leituras e outra de escritas. Dessa forma, o servidor que armazena os dados observa sempre um par de leituras e um par de escritas, e não conseguirá distinguir qual operação está sendo feita.

Na prática, ORAM vem sendo utilizada para aumentar a segurança de sistemas de busca em bases de dados. O sistema de busca proposto em [Boneh et al. 2013] sugere utilizar ORAM para recuperar os dados de registros cujos índices são conhecidos, escondendo assim informações de acesso ao servidor. Uma variante de ORAM foi proposta em [Mishra et al. 2018] para implementação de um sistema de busca indexada, cujo acesso aos índices é feito de forma oblívia. Outras aplicações de ORAM envolvendo bases de dados pode ser encontrada em [Fuller et al. 2017, Seção III.A.5]. Em geral, soluções que implementam ORAM são mais lentas, porém oferecem maior grau de segurança em situações nas quais existe risco inerente de que informações sobre padrões de acesso possam ser utilizadas, o que é sempre o caso quando bases de dados sensíveis precisam ser acessadas em servidores inseguros.

4. Ferramentas de busca e gerenciamento de dados cifrados

Sistemas que permitem buscas mais diversas em bases de dados são, em geral, baseados na combinação das técnicas citadas na Seção 3, dentre as quais algumas vazam certos tipos de informação. Além disso, cada um deles tem sua segurança avaliada considerando cenários específicos que devem ser levados em consideração quando implementados em sistemas reais, já que, na prática, informações adicionais geradas por compiladores, metadados necessários para o funcionamento do sistema, ou outros subprodutos gerados pelos sistema, são informações passíveis de serem utilizadas por atacantes.

Existe uma diversidade de sistemas de bases de dados, cada um com o seu próprio conjunto de bases primitivas que podem ser combinadas para prover suas funcionalidades. Por isso, existe a demanda de uma variedade de ferramentas de buscas protegidas para realização dessas operações primitivas de maneira segura [Fuller et al. 2017]. No contexto de base de dados, geralmente focamos em realizar pesquisas em linguagem padrão SQL. Cada operação em SQL pode ser escrita como uma combinação de operações primitivas, tais como: verificações de igualdade, comparações de ordenação e agregações. Sendo assim, podemos utilizar os esquemas de FHE ou PPEs para realizar tais operações primitivas e as compor para formar operações SQL. Aplicações baseadas em esquemas PPEs são chamadas de BoPETs (*Building on Property-revealing EncrypTion*).

Uma plataforma que utiliza PPEs em sua construção é a CryptDB [Popa et al. 2011]. Adicionalmente, na CryptDB o sistema homomórfico aditivo de [Paillier 1999] é utilizado para calcular somas. O ponto mais importante no quesito segurança é que esta biblioteca tem o princípio de cifração em camadas, e revela ao servidor somente o necessário para que o cliente seja capaz de realizar a operação SQL desejada. Quando um cliente envia determinada busca SQL, esta passa por uma etapa de pré-processamento em um servidor proxy seguro. Dependendo das comparações existentes na busca SQL , a base de dados armazenada no servidor inseguro é ajustada para a camada de cifração adequada. A CryptDB tem a vantagem de ser de fácil integração a sistemas de bases de dados preexistentes, pois as consultas feitas pelos usuário são primeiro pré-processadas por um servidor proxy e enviadas ao servidor que possui a base de

dados cifrada, sem interferir no funcionamento do gerenciador da base. Apesar de prover confidencialidade dos dados com nível de segurança controlado, a CryptDB não provê integridade dos mesmos.

Um alternativa mais atrativa no quesito segurança é baseada na utilização de busca indexada. A indexação de registros é algo comum em bases de dados que diminui os tempos de busca. Neste caso, podemos cifrar a base de dados com um esquema de cifração seguro e as pesquisas SQL são feitas diretamente nos índices, e não na própria base. Essa estrutura de índices captura a informação dos dados na base por meio de resumos criptográficos de seus valores. Funções de resumo não são invertíveis com alta probabilidade, desde que espaço de entrada da função não seja pequeno o suficiente para permitir o adversário testar todas as entradas possíveis até encontrar o resumo correspondente. Portanto, elas asseguram que, de posse desses resumos, o atacante não possa reconstruir os dados em claro.

Um dos sistemas baseados em busca indexada é o Arx [Poddar et al. 2019], que cifra os dados sensíveis no servidor com segurança semântica. Uma alternativa ao Arx é o BlindSeer [Pappas et al. 2014], um sistema que implementa busca de forma indexada, mas utilizando uma ferramenta diferente do Arx, baseada em *Bloom Filters*. A indexação é feita de forma individual, porém pode-se realizar indexação de colunas simultaneamente, com o custo de aumentar-se o armazenamento. O BlindSeer não tem suporte nativo a vários clientes, mas uma solução natural para esse problema é que os clientes compartilhem uma chave única desconhecida pelo servidor inseguro. Ao contrário do Arx, no BlindSeer as informações sobre a busca também são protegidas. Por outro lado, ele perde no quesito performance e funcionalidade em relação ao Arx.

Os sistemas acima estão sujeitos a ataques que exploram padrões de acesso, pois cada um deles apresenta vazamentos específicos que propiciam diferentes vulnerabilidades. Em tais sistemas, uma segurança indireta é a detecção da presença de um atacante online o mais rápido possível por meio de análise de tráfego malicioso [Pimenta Rodrigues et al. 2017], o que, neste caso, é facilitado pela característica específica dos padrões de acesso necessários para executar tais ataques. Existem outros sistemas com funcionalidades similares/complementares que podem ser consultados em [Fuller et al. 2017, Tabelas II e V], que inclui um sumário com a performance de cada sistema, sua segurança e funcionalidades.

Para se proteger contra ataques de padrões de acesso, vários sistemas de busca em bases de dados utilizam a técnica ORAM ou tecnologias baseadas nessa (ver p.ex. [Fuller et al. 2017, Seção III.A.5]). Eles provém segurança adicional, porém com a contrapartida de o sistema prover menos funcionalidades e/ou ter maior custo computacional. Em particular, o sistema Oblix [Mishra et al. 2018], que implementa buscas indexadas, utiliza ORAM de maneira a não somente esconder padrões de acesso do cliente acessando o servidor, como também deste quando acessando sua própria memória interna. No artigo, ele é aplicado para sistemas de busca em dados cifrados, porém também pode ser combinado com sistemas de bases de dados com pesquisas indexadas, pois o sistema de índices é dinâmico, ou seja, é passível de inserções e deleções de registros. As buscas em dados cifrados realizadas no Oblix retornam uma quantidade fixa de resultados, mesmo que existam mais ou menos resultados que satisfaçam o critério de busca. A escolha dos resultados retornados é feita através de uma métrica que quantifica o quão próximo o re-

sultado está da busca (o que é comum em aplicações de busca de palavras-chaves). Isso esconde do atacante o tamanho do conjunto de dados retornados, o que é algo atrativo no quesito segurança. Por outro lado, dependendo da aplicação em mente, o fato de existir um limite na quantidade de registros retornados em pesquisas pode ser uma limitação.

5. Ataques

Uma série de ataques práticos contra sistemas que implementam busca em bases de dados pode ter sucesso em recuperar informações em claro sobre os dados armazenados. Os objetivos são, em geral, a recuperação dos próprios dados cifrados e/ou do tipo de busca que está sendo realizada. Os ataques exploram informações de frequência, metadados armazenados no servidor, informações parcialmente conhecidas, padrões de acesso, entre outros. Em [Fuller et al. 2017, Tabela III], pode ser encontrado um sumário com tipos de ataque, seus objetivos, tempos de execução e condições necessárias para implementação. Vale ressaltar que os sistemas apresentados na seção anterior focam em resolver o problema de confidencialidade e podem não implementar integridade, como é o caso da CryptDB. Além da integridade ser uma propriedade essencial para quem utiliza os dados, sua ausência pode ser explorada por atacantes, como nos ataques realizados por administradores maliciosos de bases de dados [Akin and Sunar 2014].

Os ataques de inferência objetivam recuperar dados cifrados por meio do vazamento de informação inerente ao sistema de cifração utilizado e pelo uso adicional de informações públicas. Estes dados públicos são, em geral, provenientes de censo ou de redes sociais, que estejam correlacionados de alguma forma com a informação visada. Para avaliar a segurança de um sistema vulnerável a ataques de inferência, é necessário avaliar o impacto da utilização de informações públicas correlacionadas com os dados protegidos. Isso deve ser feito até mesmo em cenários em que o atacante obtém acesso não autorizado a bases de dados de outras instituições que estejam correlacionados de alguma forma com os dados que se pretende obter.

Em [Naveed et al. 2015] ataques reais contra sistemas de cifração que utilizam PPEs são discutidos e aplicados. Dois ataques são explorados: um baseado em informações de frequência das observações e outro na sua distribuição. Ataques de inferência são propostos de forma geral, ainda assim foram aplicados ao sistema CryptDB [Popa et al. 2011] para demonstrar sua praticidade. O ataque implementado foca em recuperar informações confidenciais em bases de dados contendo informações hospitalares e obteve sucesso significativo quando o atributo em questão possuía um número moderado de valores possíveis (p. ex. risco de mortalidade). Portanto, deve-se ter cuidado em se proteger bases de dados utilizando PPEs sem que se avalie os riscos associados. Além dos vazamentos inerentes em sistemas que utilizam PPEs, há também que se precaver de ataques que focam nos padrões de acesso existentes nas comunicações entre os clientes e o servidor que armazena a base de dados (ver Seção 3.5). Informações contidas em logs de transação quando o Arx é utilizado em conjunto com gerenciadores de bases de dados na prática podem quebrar a segurança semântica dos dados [Grubbs et al. 2017], o que exige que a gravação de logs de transação ou de outros metadados sejam desativados [Poddar et al. 2019]. Um resultado mais geral é que informações sobre padrões de acesso, quando unidas com o conhecimento da distribuição dos dados contidos na base ou com o vazamento de alguns registros, podem permitir ao atacante reconstruir boas aproximações da base de dados original [Grubbs et al. 2019].

6. Discussão

Uma das grandes dificuldades na questão da proteção de base de dados é que existem diversos tipos diferentes de sistemas. Cada um desses sistemas demanda tipos diferentes de proteção, não havendo, portanto, solução única para todos os casos. Primeiro, é necessário entender o contexto, verificar quais informações devem ser protegidas e qual o custo computacional aceitável. Mesmo assim, podem existir casos em que não tenhamos uma resposta satisfatória. De fato, as técnicas apresentadas neste trabalho nem sequer podem ser consideradas concorrentes, pois servem a propósitos distintos e, muitas vezes, podem ser utilizadas de forma complementar. De maneira geral, podemos identificar 4 tipos de usos diferentes:

- **Uso 1:** Situações em que se deseja proteger a privacidade ou anonimato dos usuários quando a base de dados for utilizada em análises por estatísticos ou outros profissionais.
- **Uso 2:** Situações em que se deseja proteger a base de dados armazenada em um servidor considerado inseguro, mas mantendo a possibilidade de a empresa (ou instituição) realizar buscas e/ou manipular dados.
- **Uso 3**: Proteção contra ataques baseados em padrões de acesso em bases de dados mantidas em um servidor inseguro e manipuladas por uma empresa (ou instituição).
- **Uso 4:** Proteção criptográfica de base de dados cujas informações de cada usuário são cifradas com chaves que estão em sua posse, protegendo a privacidade do usuário contra a própria empresa (ou instituição) que armazena ou processa seus dados. Neste contexto, cada usuário só terá acesso e poderá manipular os dados que detém.

Apesar das técnicas FHE e ORAM serem mais seguras, em alguns casos, a baixa performance acaba sendo uma característica impeditiva do seu uso na prática. Alternativas, como PPE, apresentam vulnerabilidade que as deixam sujeitas a certos tipos de ataques, porém têm potencial de aumentar a segurança de bases de dados existentes.

Conclui-se que, devido ao compromisso inerente entre segurança e funcionalidade, a escolha e o uso de tais ferramentas devem ser feitos com cautela e com o devido entendimento das limitações das ferramentas atuais. De forma geral, a utilização de tais esquemas deve sempre levar em consideração o vazamento de informação que eles possuem, avaliando quais riscos ainda serão ameaças para a instituição.

Em um cenário como esse, muitos podem ter a tentação de não utilizar nenhum tipo de defesa, porém essa não é a melhor estratégia. Em face da LGPD, empresas e órgãos que negligenciarem a proteção dos dados pessoais podem ser responsabilizados. Lembramos que segurança é um processo composto por diversos elementos que se somam, e não uma solução mágica. Assim, recomendamos o uso das técnicas de FHE e ORAM em situações nas quais seja possível aceitar o custo computacional associado. Já nos casos em que a performance é importante, recomendamos o uso de técnicas como PD ou PPEs. Em particular, como vimos na Seção 4, existem algumas ferramentas disponíveis utilizando PPEs. Apesar da potencial existência de ataques, a segurança agregada ainda é significativa, o que dificulta a recuperação de informações, bloqueando a maior parte dos adversários e contribuindo na busca da manutenção da privacidade e do cumprimento da lei.

Técnica	Anonimização	PD	FHE	PPE	ORAM
Desempenho	Alto	Alto	Baixo	Alto	Baixo
Segurança	Baixa	Média	Alta	Média	Alta
Usos	1	1	2,4	2	3

Tabela 1. Comparação entre as técnicas apresentadas neste trabalho. Deve-se destacar que as características definidas para cada técnica não refletem todos os casos. Portanto, podem existir exceções a depender da situação e do uso. Adicionalmente, algumas dessas técnicas podem ser utilizadas em conjunto para prover soluções mais robustas. Em particular, ORAM é geralmente combinada com outras técnicas quando utilizada nos protocolos de sistemas de busca em bases de dados para esconder padrões de acesso. O item segurança objetiva comparar as técnicas em relação a quantidade de informação revelada sobre o texto em claro através dos dados cifrados.

7. Conclusão

Neste trabalho foram apresentadas algumas técnicas e ferramentas de criptografia e anonimização que podem ser utilizadas na proteção de bases de dados. Pela discussão apresentada, concluímos que a escolha e implementação de tais técnicas não é uma tarefa fácil, principalmente pela inexistência de soluções padronizadas que possam ser utilizadas facilmente por diferentes instituições. Existem alternativas que possuem alto grau de segurança, como FHE e ORAM, porém o custo de performance a ser pago pode tornar o sistema inviável na prática. Outras opções, como a anonimização, PD e PPEs, são mais eficientes, mas acabam vazando certos tipos de informação, o que gera vulnerabilidades que podem ser exploradas por atacantes. Apesar das dificuldades e limitações, as técnicas de proteção de bases de dados mencionadas neste trabalho são altamente recomendadas, pois têm o potencial de aumentar significativamente a segurança dos sistemas e auxiliar no cumprimento da LGPD.

Referências

[Akin and Sunar 2014] Akin, I. H. and Sunar, B. (2014). On the difficulty of securing web applications using CryptDB. In *IEEE Fourth International Conference on Big Data and Cloud Computing*, pages 745–752.

[Alvim et al. 2011] Alvim, M. S., Andrés, M. E., Chatzikokolakis, K., and Palamidessi, C. (2011). On the relation between differential privacy and quantitative information flow. In *International Colloquium on Automata, Languages, and Programming*, pages 60–76. Springer.

[Blum et al. 2013] Blum, A., Ligett, K., and Roth, A. (2013). A learning theory approach to noninteractive database privacy. *Journal of the ACM (JACM)*, 60(2):1–25.

[Boneh et al. 2013] Boneh, D., Gentry, C., Halevi, S., Wang, F., and Wu, D. J. (2013). Private database queries using somewhat homomorphic encryption. In *International Conference on Applied Cryptography and Network Security*, pages 102–118. Springer.

[Boneh et al. 2011] Boneh, D., Sahai, A., and Waters, B. (2011). Functional encryption: Definitions and challenges. In *Theory of Cryptography Conference*, pages 253–273. Springer.

[Boneh and Shoup 2015] Boneh, D. and Shoup, V. (2015). A graduate course in applied cryptography. *Draft 0.2*.

[Bourse et al. 2018] Bourse, F., Minelli, M., Minihold, M., and Paillier, P. (2018). Fast homomorphic evaluation of deep discretized neural networks. In *Annual International Cryptology Conference*, pages 483–512. Springer.

[Brightwell and Smith 1997] Brightwell, M. and Smith, H. (1997). Using datatype-preserving encryption to enhance data warehouse security. In *20th National Information Systems Security Conference Proceedings (NISSC)*.

[Chillotti et al. 2016] Chillotti, I., Gama, N., Georgieva, M., and Izabachène, M. (2016). TFHE: Fast fully homomorphic encryption library. https://tfhe.github.io/tfhe/.

[Dai and Sunar 2015] Dai, W. and Sunar, B. (2015). cuhe: A homomorphic encryption accelerator library. In *International Conference on Cryptography and Information Security in the Balkans*, pages 169–186. Springer.

[Ducas and Micciancio 2015] Ducas, L. and Micciancio, D. (2015). FHEW: bootstrapping homomorphic encryption in less than a second. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 617–640. Springer.

[Durak 2017] Durak, F. (2017). *Cryptanalytic study of property-preserving encryption*. PhD thesis, Rutgers University-School of Graduate Studies, New Brunswick, NJ, USA.

[Dwork 2008] Dwork, C. (2008). Differential privacy: A survey of results. In *International conference on theory and applications of models of computation*, pages 1–19. Springer.

[Dwork and Nissim 2004] Dwork, C. and Nissim, K. (2004). Privacy-preserving datamining on vertically partitioned databases. In *Annual International Cryptology Conference*, pages 528–544. Springer.

[Dwork et al. 2014] Dwork, C., Roth, A., et al. (2014). The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407.

[Fuller et al. 2017] Fuller, B., Varia, M., Yerukhimovich, A., Shen, E., and Hamlin, A. (2017). SoK : Cryptographically Protected Database Search. *IEEE Symposium on Security and Privacy (SP)*, pages 172–191.

[Gentry et al. 2012] Gentry, C., Halevi, S., and Smart, N. P. (2012). Homomorphic evaluation of the AES circuit. In *Annual Cryptology Conference*, pages 850–867. Springer.

[Goldreich and Ostrovsky 1996] Goldreich, O. and Ostrovsky, R. (1996). Software protection and simulation on oblivious rams. *Journal of the ACM (JACM)*, 43(3):431–473.

[Grubbs et al. 2019] Grubbs, P., Lacharité, M.-S., Minaud, B., and Paterson, K. G. (2019). Learning to reconstruct: Statistical learning theory and encrypted database attacks. In *IEEE Symposium on Security and Privacy (SP)*, pages 1067–1083.

[Grubbs et al. 2017] Grubbs, P., Ristenpart, T., and Shmatikov, V. (2017). Why your encrypted database is not secure. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, pages 162–168.

[Jain et al. 2016] Jain, P., Gyanchandani, M., and Khare, N. (2016). Big data privacy: a technological perspective and review. *Journal of Big Data*, 3(1):25.

[Li et al. 2007] Li, N., Li, T., and Venkatasubramanian, S. (2007). t-closeness: Privacy beyond k-anonymity and l-diversity. In *IEEE 23rd International Conference on Data Engineering*, pages 106–115.

[Machanavajjhala et al. 2007] Machanavajjhala, A., Kifer, D., Gehrke, J., and Venkitasubramaniam, M. (2007). l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):3–es.

[Mattsson and Rozenberg 2013] Mattsson, U. and Rozenberg, Y. (2013). Tokenization in payment environments. US Patent App. 13/761,009.

[Mishra et al. 2018] Mishra, P., Poddar, R., Chen, J., Chiesa, A., and Popa, R. A. (2018). Oblix: An efficient oblivious search index. In *IEEE Symposium on Security and Privacy (SP)*, pages 279–296.

[Narayanan and Shmatikov 2008] Narayanan, A. and Shmatikov, V. (2008). Robust deanonymization of large sparse datasets. In *IEEE Symposium on Security and Privacy*, pages 111–125.

[Naveed et al. 2015] Naveed, M., Kamara, S., and Wright, C. V. (2015). Inference attacks on property-preserving encrypted databases. *Proceedings of the ACM Conference on Computer and Communications Security*, 2015:644–655.

[Paillier 1999] Paillier, P. (1999). Public-Key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*, pages 223–238. Springer.

[Pappas et al. 2014] Pappas, V., Krell, F., Vo, B., Kolesnikov, V., Malkin, T., Choi, S. G., George, W., Keromytis, A., and Bellovin, S. (2014). Blind Seer: A scalable private DBMS. In *IEEE Symposium on Security and Privacy*, pages 359–374.

[Pimenta Rodrigues et al. 2017] Pimenta Rodrigues, G. A., de Oliveira Albuquerque, R., Gomes de Deus, F. E., De Oliveira Júnior, G. A., García Villalba, L. J., Kim, T.-H., et al. (2017). Cybersecurity and network forensics: Analysis of malicious traffic towards a honeynet with deep packet inspection. *Applied Sciences*, 7(10):1082.

[Poddar et al. 2019] Poddar, R., Boelter, T., and Popa, R. A. (2019). Arx: an encrypted database using semantically secure encryption. *Proceedings of the VLDB Endowment*, 12(11):1664–1678.

[Popa et al. 2011] Popa, R. A., Redfield, C. M., Zeldovich, N., and Balakrishnan, H. (2011). CryptDB: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 85–100.

[Stalla-Bourdillon and Knight 2016] Stalla-Bourdillon, S. and Knight, A. (2016). Anonymous data v. personal data-false debate: An eu perspective on anonymization, pseudonymization and personal data. *Wis. Int'l LJ*, 34:284.

[Sweeney 2002] Sweeney, L. (2002). k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570.

Improving the Security of ChaCha against Differential-Linear Cryptanalysis

Murilo Coutinho [1,2], **Iago Passos** [2], **Rafael T. de Sousa Jr.** [2], **Fábio Borges** [3]

[1]Centro de Pesquisa e Desenvolvimento para a Segurança da Comunicações (CEPESC)
Agência Brasileira de Inteligência
Brasília, Brasil.

[2]Departamento de Engenharia Elétrica
Universidade de Brasília (UnB)
Brasília, Brasil.

[3]Laboratório Nacional de Computação Científica (LNCC)
Petrópolis, Brasil

Abstract. *The stream cipher ChaCha has received a lot of attention and recently is being used as a new cipher suite in TLS 1.3, as a random number generator for operating systems (Linux, FreeBSD, OpenBSD, NetBSD, and DragonFly BSD), a proposed standardization in RFC 7634 for use IKE and IPsec, and by the WireGuard VPN protocol. Because of that, it is very important to understand and study the security of this algorithm. Previous works showed that it is possible to break up to 7 of the 20 rounds of ChaCha. In this paper, we show that a simple modification in the algorithm, namely changing the rotation distances in the Quarter Round Function, makes ChaCha more secure against all the most effective known attacks without any loss in performance. In fact, we show that with these changes, it is only possible to break up to 6 rounds of ChaCha. Therefore, it would be no longer possible to break 7 rounds of ChaCha with the best-known attacks.*

1. Introduction

In 2008, Bernstein proposed the stream cipher Salsa20 [Bernstein 2008b] as a contender to the eStream competition. Later, Bernstein proposed some modifications to Salsa20 to improve diffusion and security, creating a new stream cipher, which he called ChaCha20 [Bernstein 2008a]. Although Salsa20 was one of the winners of the eStream competition, ChaCha20 has received much more attention through the years. Nowadays, we see the usage of this cipher in several projects and applications.

ChaCha, along Poly1305 [Bernstein 2005], is in one of the cipher suits of the new TLS 1.3 [Langley et al. 2016], which is actually used by Google on both Chrome and Android. ChaCha is used not only in TLS but in many other protocols such as SSH, Noise, and S/MIME 4.0. In addition, the RFC 7634 proposes the use of ChaCha in IKE and IPsec. ChaCha is used not only for encryption but also as a random number generator, for example, in any operating system running Linux kernel 4.8 or newer [Torvalds 2016]. Additionally, ChaCha is used in several applications, for example, WireGuard (VPN), Keepass (password manager), and Veracrypt (disk encryption). See [IANIX 2020] for a huge list of applications, protocols, and libraries using ChaCha20.

Since ChaCha is so heavily used, it is very important to fully understand its security. Indeed, the cryptanalysis of ChaCha is well understood and several authors studied its security [Aumasson et al. 2008, Hernandez-Castro et al. 2008, Crowley 2006, Fischer et al. 2006, Ishiguro et al. 2011, Maitra 2016, Maitra et al. 2015, Mouha and Preneel 2013, Choudhuri and Maitra 2016, Shi et al. 2012, Tsunoo et al. 2007, Dey and Sarkar 2017, Dey et al. 2019, Ding 2019, Coutinho and Neto 2020].

In this work, we study the most important attacks against ChaCha and show that it is possible to improve its security by changing the rotation distances in the Quarter Round Function (QRF). In fact, to this day, the best attack against ChaCha works on only 7 rounds of the 20 provided by the algorithm. However, using the proposed modification, we show that the security is enhanced, limiting the best attack to succeed on only 6 rounds.

This work is organized as follows: in Section 2, we define the notation used in the paper and define the ChaCha algorithm. In Section 3, we review the best attacks available against ChaCha. In Section 4, we provide an intensive analysis of the security of the algorithm for all combinations of rotation distances showing that it is possible to improve the security of ChaCha. In Section 5, we provide a security comparison of the original ChaCha, and its new proposed version. Finally, in Section 6, we present the conclusions.

2. Specifications and Preliminaries

In this section, we define the notation that we will use throughout the paper in Table 1. Afterwards, we define the algorithm ChaCha.

Bernstein proposed the stream cipher Salsa [Bernstein 2008b] to the *eStream* competition and later Bernstein proposed ChaCha [Bernstein 2008a] as an improvement of Salsa. ChaCha consists of a series of ARX (addition, rotation, and XOR) operations on 32-bit words, being highly efficient in software and hardware. Salsa operates on a state of 64 bytes, organized as a 4×4 matrix with 32-bit integers, initialized with a 256-bit key $k_0, k_1, ..., k_7$, a 64-bit nonce v_0, v_1 and a 64-bit counter t_0, t_1 (we may also refer to the nonce and counter words as the initialization vector – IV), and 4 constants $c_0 = 0\text{x}61707865$, $c_1 = 0\text{x}3320646e$, $c_2 = 0\text{x}79622d32$ and $c_3 = 0\text{x}6b206574$. For ChaCha, we have the following initial state matrix:

$$X^{(0)} = \begin{pmatrix} x_0^{(0)} & x_1^{(0)} & x_2^{(0)} & x_3^{(0)} \\ x_4^{(0)} & x_5^{(0)} & x_6^{(0)} & x_7^{(0)} \\ x_8^{(0)} & x_9^{(0)} & x_{10}^{(0)} & x_{11}^{(0)} \\ x_{12}^{(0)} & x_{13}^{(0)} & x_{14}^{(0)} & x_{15}^{(0)} \end{pmatrix} = \begin{pmatrix} c_0 & c_1 & c_2 & c_3 \\ k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ t_0 & t_1 & v_0 & v_1 \end{pmatrix}. \tag{1}$$

The state matrix is modified in each round by a *Quarter Round Function* (QRF), named $QR_{r_1,r_2,r_3,r_4}(a, b, c, d)$, which receives and updates 4 integers in the following way:

$$\begin{array}{lllllllll} a & \leftarrow & a+b; & d & \leftarrow & d \oplus a; & d & \leftarrow & d \lll r_1; \\ c & \leftarrow & c+d; & b & \leftarrow & b \oplus c; & b & \leftarrow & b \lll r_2; \\ a & \leftarrow & a+b; & d & \leftarrow & d \oplus a; & d & \leftarrow & d \lll r_3; \\ c & \leftarrow & c+d; & b & \leftarrow & b \oplus c; & b & \leftarrow & b \lll r_4; \end{array} \tag{2}$$

Notation	Description
X	a 4×4 state matrix of the cipher of 16 words
$X^{(0)}$	initial state matrix
$X^{(R)}$	state matrix after application of R round functions
Z	output of an algorithm, $Z = X + X^{(R)}$
$x_i^{(R)}$	i^{th} word of the state matrix $X^{(R)}$ (words arranged in row major)
$x_{i,j}^{(R)}$	j^{th} bit of ith word of the state matrix $X^{(R)}$
$x + y$	addition of x and y modulo 2^{32}
$x - y$	subtraction of x and y modulo 2^{32}
$x \oplus y$	bitwise XOR of x and y
$x \lll n$	rotation of x by n bits to the left
$x \ggg n$	rotation of x by n bits to the right
Δx	XOR difference of x and x'. $\Delta x = x \oplus x'$
$\Delta_i^{(R)}$	differential $\Delta_i^{(R)} = x_i^{(R)} \oplus {x'}_i^{(R)}$
$\Delta_{i,j}^{(R)}$	differential $\Delta_{i,j}^{(R)} = x_{i,j}^{(R)} \oplus {x'}_{i,j}^{(R)}$
$\mathbb{P}(E)$	probability of occurrence of an event E
$\mathbb{B}(E)$	bias of an event E, thus $\mathbb{B}(E) = 2\mathbb{P}(E) - 1$
$\varepsilon_{(x_1 \oplus ... \oplus x_m)}$	bias of event $E = \{\Delta x_1 \oplus ... \oplus \Delta x_m = 0\}$
$\mathcal{I}D$	input differential
$\mathcal{O}D$	output differential

Table 1. Notation.

One round of ChaCha is defined as 4 applications of $QR_{16,12,8,7}$. There is a difference, however, between odd and even rounds. For odd rounds, when $r \in \{1, 3, 5, 7, ...\}$, $X^{(r)}$ is defined from $X^{(r-1)}$, as

$$\begin{array}{rcl}
\left(x_0^{(r)}, x_4^{(r)}, x_8^{(r)}, x_{12}^{(r)}\right) & \leftarrow & QR_{16,12,8,7}\left(x_0^{(r-1)}, x_4^{(r-1)}, x_8^{(r-1)}, x_{12}^{(r-1)}\right) \\
\left(x_1^{(r)}, x_5^{(r)}, x_9^{(r)}, x_{13}^{(r)}\right) & \leftarrow & QR_{16,12,8,7}\left(x_1^{(r-1)}, x_5^{(r-1)}, x_9^{(r-1)}, x_{13}^{(r-1)}\right) \\
\left(x_2^{(r)}, x_6^{(r)}, x_{10}^{(r)}, x_{14}^{(r)}\right) & \leftarrow & QR_{16,12,8,7}\left(x_2^{(r-1)}, x_6^{(r-1)}, x_{10}^{(r-1)}, x_{14}^{(r-1)}\right) \\
\left(x_3^{(r)}, x_7^{(r)}, x_{11}^{(r)}, x_{15}^{(r)}\right) & \leftarrow & QR_{16,12,8,7}\left(x_3^{(r-1)}, x_7^{(r-1)}, x_{11}^{(r-1)}, x_{15}^{(r-1)}\right)
\end{array},$$

and, for even rounds $r \in \{2, 4, 6, 8, ...\}$, as

$$\begin{array}{rcl}
\left(x_0^{(r)}, x_5^{(r)}, x_{10}^{(r)}, x_{15}^{(r)}\right) & \leftarrow & QR_{16,12,8,7}\left(x_0^{(r-1)}, x_5^{(r-1)}, x_{10}^{(r-1)}, x_{15}^{(r-1)}\right) \\
\left(x_1^{(r)}, x_6^{(r)}, x_{11}^{(r)}, x_{12}^{(r)}\right) & \leftarrow & QR_{16,12,8,7}\left(x_1^{(r-1)}, x_6^{(r-1)}, x_{11}^{(r-1)}, x_{12}^{(r-1)}\right) \\
\left(x_2^{(r)}, x_7^{(r)}, x_8^{(r)}, x_{13}^{(r)}\right) & \leftarrow & QR_{16,12,8,7}\left(x_2^{(r-1)}, x_7^{(r-1)}, x_8^{(r-1)}, x_{13}^{(r-1)}\right) \\
\left(x_3^{(r)}, x_4^{(r)}, x_9^{(r)}, x_{14}^{(r)}\right) & \leftarrow & QR_{16,12,8,7}\left(x_3^{(r-1)}, x_4^{(r-1)}, x_9^{(r-1)}, x_{14}^{(r-1)}\right)
\end{array}.$$

The algorithm ChaCha20/R is then defined as the sum of the initial state with the state obtained after R rounds of operations $Z = X + X^{(R)}$. One should note that it is possible to parallelize each application of the QRF on each round and that each round is reversible. Hence, we can compute $X^{(r-1)}$ from $X^{(r)}$. For more information on ChaCha, we refer to [Bernstein 2008a].

3. Cryptanalysis of ChaCha

Several authors studied the security and diffusion of both Salsa and ChaCha [Aumasson et al. 2008, Hernandez-Castro et al. 2008, Crowley 2006, Coutinho et al. 2020, Fischer et al. 2006, Ishiguro et al. 2011, Maitra 2016, Maitra et al. 2015, Mouha and Preneel 2013, Choudhuri and Maitra 2016, Shi et al. 2012, Tsunoo et al. 2007, Dey and Sarkar 2017, Dey et al. 2019, Ding 2019, Coutinho and Neto 2020] which show weaknesses in the reduced rounds of the ciphers. The attacks in most cases, apply some input differences to the initial state to observe output differences after certain rounds. Once one of them can proceed a few rounds forward as above, it may be possible to invert a few rounds from a final state to obtain further non-randomness. Crowley introduced the cryptanalysis of Salsa [Crowley 2006] in 2006, but the most important cryptanalysis in this regard was proposed by Aumasson et al. at FSE 2008 [Aumasson et al. 2008] with the introduction of Probabilistic Neutral Bits (PNBs). After that, several authors proposed small enhancements on the attack of Aumasson et al. The work by Shi et al [Shi et al. 2012] introduced the concept of Column Chaining Distinguisher (CCD) to achieve some incremental advancements over [Aumasson et al. 2008] for both Salsa and ChaCha. Maitra, Paul, and Meier [Maitra et al. 2015] studied an interesting observation about round reversal of Salsa, but no significant cryptanalytic improvement could be obtained using this method. Maitra [Maitra 2016] used a technique of Chosen IVs to obtain certain improvements over existing results. Dey and Sarkar [Dey and Sarkar 2017] showed how to choose values for the PNB to improve the attack. The best improvement for the technique was given by Choudhuri and Maitra [Choudhuri and Maitra 2016] using the technique of differential-linear cryptanalysis and exploring the mathematical structure of both Salsa and ChaCha to find differential characteristics with much higher biases. Later Coutinho and Neto improved Choudhuri and Maitra's attack by showing better linear approximations [Coutinho and Neto 2020].

Here, we analyze and improve the security of ChaCha by first replicating and checking the results of the attack of Aumasson [Aumasson et al. 2008], Choudhuri and Maitra [Choudhuri and Maitra 2016], and Coutinho and Neto [Coutinho and Neto 2020] and then applying the technique against different rotation combinations for the QRF. We chose these attacks since they are the most important works on the cryptanalysis of Salsa and ChaCha to this day.

3.1. Probabilistic Neutral Bits

This section reviews the attack of Aumasson [Aumasson et al. 2008]. The attack first identifies good choices of truncated differentials, then it uses probabilistic backwards computation with the notion of Probabilistic Neutral Bits (PNB), and, finally, it estimates the complexity of the attack.

Let Δ_i^R be the differential for the i_{th} word of state matrix $X^{(R)}$, thus $\Delta_i^R = x_i^R \oplus {x'}_i^R$; and let $\Delta_{i,j}^R$ be the differential for the j_{th} bit of the i_{th} word, thus $\Delta_{i,j}^R = x_{i,j}^R \oplus {x'}_{i,j}^R$. In [Aumasson et al. 2008] the input differential $\mathcal{I}D$ is defined for a single-bit difference $\Delta_{i,j}^0 = 1$ and consider a single-bit output difference $\mathcal{O}D$ after r rounds $\Delta_{p,q}^r$, such differential is denoted $(\Delta_{p,q}^r | \Delta_{i,j}^0)$. For a fixed key, the bias ε_d of the $\mathcal{O}D$ is defined by $\mathbb{P}_{v,t}(\Delta_{p,q}^r = 1 | \Delta_{i,j}^0) = \frac{1}{2}(1 + \varepsilon_d)$, where the probability holds over all nonces v and coun-

ters t. Furthermore, considering the key as a random variable, we denote the median value of ε_d by $\varepsilon_d^\star$. Hence, for half of the keys, this differential have a bias of at least $\varepsilon_d^\star$.

Now, assume that the differential $(\Delta_{p,q}^r|\Delta_{i,j}^0)$ of bias ε_d is fixed, and we observe outputs Z and Z' of $R = l + r$ rounds for nonce v, counter t and unknown key k. If we guess the key k we can invert l rounds of the algorithm to get $X^{(r)}$ and $X'^{(r)}$ and compute $\Delta_{p,q}^r$, let f be the function which executes this procedure. Hence $f(k, v, t, Z, Z') = \Delta_{p,q}^r$ and we expect that

$$\mathbb{P}(f(\hat{k}, v, t, Z, Z') = 1) = \begin{cases} \frac{1}{2}(1 + \varepsilon_d), & \text{if } \hat{k} = k \\ 0.5, & \text{if } \hat{k} \neq k \end{cases},$$

thus, if we have several pairs of Z and Z', it is possible to test our guesses for k.

Thus, we can search only over a subkey of $m = 256 - n$ bits, provided we can find a function g that approximates f but only uses m key bits as input. Then, let $\bar{k}$ correspond to the subkey of m bits of key k and let f to be correlated to g with bias ε_a i.e., $\mathbb{P}(f(k, v, t, Z, Z') = g(\bar{k}, v, t, Z, Z')) = \frac{1}{2}(1 + \varepsilon_a)$.

If we denote the bias of g by ε, i.e. $P(g(\bar{k}, v, t, Z, Z') = 1) = \frac{1}{2}(1 + \varepsilon)$, and $\varepsilon^\star$ the median bias of g over all keys, we can approximate ε by $\varepsilon_d\varepsilon_a$. The problem that remains is how to efficiently find such a function g. In [Aumasson et al. 2008], this is done by first identifying key bits that have little influence on the result of $f(k, v, t, Z, Z')$, these are called *probabilistic neutral bits* (PNBs). This is done by defining the *neutrality measure* $\gamma_{i,j}$ of a key bit $k_{i,j}$.

After computing $\gamma_{i,j}$ (see [Aumasson et al. 2008] for a method of estimation), for all $i = (0, 1, ..., 7)$ and $j = (0, 1, ..., 31)$, we can define the set of significant key bits as $\Psi = \{(i, j) : \gamma_{i,j} \leq \gamma\}$ where γ is a threshold value, and then define our approximation g as $g(k_\Psi, v, t, Z, Z') = f(k^*, v, t, Z, Z')$ where k_Ψ is defined as the subkey with key bits in the set Ψ and k^* is computed from k_Ψ by setting $k_{i,j} = 0$ for all $(i, j) \notin \Psi$. Thus, the attack can be evaluated with the following steps:

1. Compute a good differential for r rounds $(\Delta_{p,q}^r|\Delta_{i,j}^0)$ by estimating the bias ε_d for all single-bit $\mathcal{I}D$ with several random combinations of keys, nonces, and counters.
2. Empirically estimate the neutrality measure $\gamma_{r,s}$ for each key bit $k_{r,s}$.
3. Construct the function g by setting all key bit such that $\gamma_{r,s} > \gamma$ to zero and estimate the median bias $\varepsilon^\star$ by empirically measuring bias of g using many randomly chosen keys, nonces, and counters.
4. Estimate the data and time complexity of the attack.

We refer to [Aumasson et al. 2008] for further information about the estimation of the data and time complexity of the attack and for further details on the described technique.

3.2. Multi-bit Differentials

This section reviews the attack of Choudhuri and Maitra [Choudhuri and Maitra 2016] and later improved by Coutinho and Neto [Coutinho and Neto 2020]. The attack first identifies linear relationships between the bits of two successive rounds of ChaCha. From these relationships, it is possible to compute single bit differentials for r rounds, obtaining

a distinguisher for $r+1$ rounds, which reduces the complexities of the attacks described in Section 3.1. The first step is to write ChaCha's QRF (Eq. (2)) only using XOR bit by bit operations.

Then, is possible to compute $x_{a,i}^{(m-1)}, x_{b,i}^{(m-1)}, x_{c,i}^{(m-1)}$, and $x_{d,i}^{(m-1)}$ in terms of $x_{a,i}^{(m)}, x_{b,i}^{(m)}, x_{c,i}^{(m)}, x_{d,i}^{(m)}, C_i^1, C_i^2, C_i^3$, and C_i^4, we get:

$$\begin{array}{rcl} x_{a,i}^{(m-1)} & = & x_{a,i}^{(m)} \oplus x_{b,i+r_4}^{(m)} \oplus x_{b,i+r_2+r_4}^{(m)} \oplus x_{c,i+r_2}^{(m)} \oplus x_{d,i}^{(m)} \oplus C_i^4 \oplus C_i^3 \oplus C_i^1 \\ x_{b,i}^{(m-1)} & = & x_{b,i+r_2+r_4}^{(m)} \oplus x_{c,i+r_2}^{(m)} \oplus x_{d,i}^{(m)} \oplus x_{c,i}^{(m)} \oplus C_i^4 \\ x_{c,i}^{(m-1)} & = & x_{d,i}^{(m)} \oplus x_{c,i}^{(m)} \oplus x_{d,i+r_3}^{(m)} \oplus x_{a,i}^{(m)} \oplus C_i^2 \oplus C_i^4 \\ x_{d,i}^{(m-1)} & = & x_{d,i+r_1+r_3}^{(m)} \oplus x_{a,i+r_1}^{(m)} \oplus x_{a,i}^{(m)} \oplus x_{c,i}^{(m)} \oplus x_{b,i+r_4}^{(m)} \oplus C_i^3 \end{array} \tag{3}$$

where C_i^1, C_i^2, C_i^3, and C_i^4 denote the i-th carry bit of the first, second, third, and fourth additions contained in the QRF, respectively. Since we have that $C_0^1 = C_0^2 = C_0^3 = C_0^4 = 0$ by definition, the following lemma has been proved in [Choudhuri and Maitra 2016]:

Lemma 1. *Let*

$$\begin{array}{rcl} \Delta A^{(m)} & = & \Delta x_{\alpha,0}^{(m)} \oplus \Delta x_{\beta,r_4}^{(m)} \oplus \Delta x_{\beta,r_2+r_4}^{(m)} \oplus \Delta x_{\gamma,r_2}^{(m)} \oplus \Delta x_{\delta,0}^{(m)} \\ \Delta B^{(m)} & = & \Delta x_{\beta,r_2+r_4}^{(m)} \oplus \Delta x_{\gamma,0}^{(m)} \oplus \Delta x_{\gamma,r_2}^{(m)} \oplus \Delta x_{\delta,0}^{(m)} \\ \Delta C^{(m)} & = & \Delta x_{\delta,0}^{(m)} \oplus \Delta x_{\gamma,0}^{(m)} \oplus \Delta x_{\delta,r_3}^{(m)} \oplus \Delta x_{\alpha,0}^{(m)} \\ \Delta D^{(m)} & = & \Delta x_{\delta,r_1+r_3}^{(m)} \oplus \Delta x_{\alpha,r_1}^{(m)} \oplus \Delta x_{\alpha,0}^{(m)} \oplus \Delta x_{\gamma,0}^{(m)} \oplus \Delta x_{\beta,r_4}^{(m)} \end{array}$$

After m rounds of ChaCha, the following holds:

$$\left|\varepsilon_{(A^{(m)})}\right| = \left|\varepsilon_{x_{\alpha,0}^{(m-1)}}\right|, \quad \left|\varepsilon_{(B^{(m)})}\right| = \left|\varepsilon_{x_{\beta,0}^{(m-1)}}\right|,$$

$$\left|\varepsilon_{(C^{(m)})}\right| = \left|\varepsilon_{x_{\gamma,0}^{(m-1)}}\right|, \quad \left|\varepsilon_{(D^{(m)})}\right| = \left|\varepsilon_{x_{\delta,0}^{(m-1)}}\right|,$$

The tuples $(\alpha, \beta, \gamma, \delta)$ vary depending on whether m is odd or even

1. *m is odd:* $(\alpha, \beta, \gamma, \delta) \in \{(0,4,8,12), (1,5,9,13), (2,6,10,14), (3,7,11,15)\}$
2. *m is even:* $(\alpha, \beta, \gamma, \delta) \in \{(0,5,10,15), (1,6,11,12), (2,7,8,13), (3,4,9,14)\}$

Using Lemma 1, it is possible to find differentials for $m-1$ rounds and then use the bias to make a distinguisher for a linear combination in m rounds, improving the attack. It is possible to go one round further by using linear equations that hold with high probability. The following lemma defines useful linear relationships, see [Choudhuri and Maitra 2016] for a proof of this lemma.

Lemma 2. *For ChaCha, each of the following holds with probability $\frac{1}{2}(1+\frac{1}{2})$:*

$$\begin{array}{rl} x_{8,0}^{(3)} = & x_{13,r_1+r_3}^{(5)} \oplus x_{1,r_1}^{(5)} \oplus x_{1,0}^{(5)} \oplus x_{9,0}^{(5)} \oplus x_{5,r_4}^{(5)} \oplus x_{12,0}^{(5)} \oplus x_{8,0}^{(5)} \oplus \\ & x_{12,r_3}^{(5)} \oplus x_{0,0}^{(5)} \oplus x_{2,0}^{(5)} \oplus x_{6,r_4}^{(5)} \oplus x_{6,r_2+r_4}^{(5)} \oplus x_{10,r_2}^{(5)} \oplus x_{14,0}^{(5)} \oplus \\ & x_{13,2r_3+r1}^{(5)} \oplus x_{1,r_1+r_3}^{(5)} \oplus x_{1,r_3}^{(5)} \oplus x_{9,r_3}^{(5)} \oplus x_{5,r_3+r_4}^{(5)} \oplus x_{5,r_3+r_4-1}^{(5)} \oplus x_{9,r_3-1}^{(5)} \end{array}$$

$$\begin{array}{rl} x_{9,0}^{(3)} = & x_{14,r_1+r_3}^{(5)} \oplus x_{2,r_1}^{(5)} \oplus x_{2,0}^{(5)} \oplus x_{10,0}^{(5)} \oplus x_{6,r_4}^{(5)} \oplus x_{13,0}^{(5)} \oplus x_{9,0}^{(5)} \oplus \\ & x_{13,r_3}^{(5)} \oplus x_{1,0}^{(5)} \oplus x_{3,0}^{(5)} \oplus x_{7,r_4}^{(5)} \oplus x_{7,r_2+r_4}^{(5)} \oplus x_{11,r_2}^{(5)} \oplus x_{15,0}^{(5)} \oplus \\ & x_{14,2r_3+r1}^{(5)} \oplus x_{2,r_1+r_3}^{(5)} \oplus x_{2,r_3}^{(5)} \oplus x_{10,r_3}^{(5)} \oplus x_{6,r_3+r_4}^{(5)} \oplus x_{6,r_3+r_4-1}^{(5)} \oplus x_{10,r_3-1}^{(5)} \end{array}$$

$$
\begin{aligned}
x^{(3)}_{10,0} = \; & x^{(5)}_{15,r_1+r_3} \oplus x^{(5)}_{3,r_1} \oplus x^{(5)}_{3,0} \oplus x^{(5)}_{11,0} \oplus x^{(5)}_{7,r_4} \oplus x^{(5)}_{14,0} \oplus x^{(5)}_{10,0} \oplus \\
& x^{(5)}_{14,r_3} \oplus x^{(5)}_{2,0} \oplus x^{(5)}_{0,0} \oplus x^{(5)}_{4,r_4} \oplus x^{(5)}_{4,r_2+r_4} \oplus x^{(5)}_{8,r_2} \oplus x^{(5)}_{12,0} \oplus \\
& x^{(5)}_{15,2r_3+r1} \oplus x^{(5)}_{3,r_1+r_3} \oplus x^{(5)}_{3,r_3} \oplus x^{(5)}_{11,r_3} \oplus x^{(5)}_{7,r_3+r_4} \oplus x^{(5)}_{7,r_3+r_4-1} \oplus x^{(5)}_{11,r_3-1} \\
x^{(3)}_{11,0} = \; & x^{(5)}_{12,r_1+r_3} \oplus x^{(5)}_{0,r_1} \oplus x^{(5)}_{0,0} \oplus x^{(5)}_{8,0} \oplus x^{(5)}_{4,r_4} \oplus x^{(5)}_{15,0} \oplus x^{(5)}_{11,0} \oplus \\
& x^{(5)}_{15,r_3} \oplus x^{(5)}_{3,0} \oplus x^{(5)}_{1,0} \oplus x^{(5)}_{5,r_4} \oplus x^{(5)}_{5,r_2+r_4} \oplus x^{(5)}_{9,r_2} \oplus x^{(5)}_{13,0} \oplus \\
& x^{(5)}_{12,2r_3+r1} \oplus x^{(5)}_{0,r_1+r_3} \oplus x^{(5)}_{0,r_3} \oplus x^{(5)}_{8,r_3} \oplus x^{(5)}_{4,r_3+r_4} \oplus x^{(5)}_{4,r_3+r_4-1} \oplus x^{(5)}_{8,r_3-1}
\end{aligned}
$$

Latter Coutinho and Neto showed the following Lemmas [Coutinho and Neto 2020], which they used to significantly improve the attacks against ChaCha

Lemma 3. *Let*

$$\Delta E^{(m)} = \Delta x^{(m)}_{\alpha,0} \oplus \Delta x^{(m)}_{\beta,r_4} \oplus \Delta x^{(m)}_{\gamma,0}$$

After m rounds of ChaCha, the following holds:

$$\left|\varepsilon_{(E^{(m)})}\right| = \left|\varepsilon_{(x^{(m-1)}_{\alpha,0} \oplus x^{(m-1)}_{\beta,0})}\right|$$

The tuples (α, β, γ) vary depending on whether m is odd or even.

- *Case I. m odd:* $(\alpha, \beta, \gamma) \in \{(0,4,8), (1,5,9), (2,6,10), (3,7,11)\}$
- *Case II. m even:* $(\alpha, \beta, \gamma) \in \{(0,5,10), (1,6,11), (2,7,8), (3,4,9)\}$

Lemma 4. *When m is odd, the following holds with probability $\frac{1}{2}(1+\frac{1}{2})$*

$$
\begin{aligned}
x^{(m-2)}_{3,0} \oplus x^{(m-2)}_{4,0} = \; & x^{(m)}_{1,0} \oplus x^{(m)}_{3,0} \oplus x^{(m)}_{4,2r_4+r_2} \oplus x^{(m)}_{7,r_4} \oplus x^{(m)}_{7,r_2+r_4} \oplus \\
& x^{(m)}_{8,r_4} \oplus x^{(m)}_{8,r_2+r_4} \oplus x^{(m)}_{9,0} \oplus x^{(m)}_{11,r_2} \oplus x^{(m)}_{12,r_4-1} \oplus \\
& x^{(m)}_{12,r_4} \oplus x^{(m)}_{13,0} \oplus x^{(m)}_{13,r_3} \oplus x^{(m)}_{15,0}.
\end{aligned}
$$

4. Improving ChaCha

In [Bernstein 2008a], Bernstein justify the choice of the rotation distances 16, 12, 8, 7 with the argument:

> "The above code also shows a much less important difference between ChaCha and Salsa20: I changed the rotation distances 7, 9, 13, 18 to 16, 12, 8, 7. The difference in security appears to be negligible: 7, 9, 13, 18 appears marginally better in some diffusion measures, and 16, 12, 8, 7 appears marginally better in others, but the margins are tiny, far smaller than the presumed inaccuracy of the diffusion measures as predictors of security. The change boosts speed slightly on some platforms while making no difference on other platforms".

Naturally, the attacks against ChaCha were unknown by the time of its publication. Therefore, one might expect that there could exist a distinct set of rotation distances such that ChaCha has better security against differential and linear cryptanalysis. Thus, our approach to improve the security of ChaCha consists in testing all combination of rotation distances to find if there is a set that is more secure.

4.1. Testing Differential Paths

In [Aumasson et al. 2008], the authors presented attacks for 6 and 7 rounds of ChaCha, however, both attacks use differential paths for $r = 3$ rounds. Leveraging this fact, our first test consists in computing the best differential path for 3 rounds of ChaCha considering all single-bit input differentials and all output bits. In other words, we estimated the bias ε_d for all combinations of differentials $(\Delta^r_{p,q}|\Delta^0_{i,j})$ for each combination of rotations distances. Since each rotation have 32 values and since we have 128 input differentials and 512 output bits, we conclude that we computed $128 \times 512 \times 32^4 = 2^{36}$ different biases.

More specifically, we used Algorithm 1 to compute the highest bias for all combinations of rotations distances. Unfortunately, since we are performing an empirical estimation, we need to execute the same procedure several times for each input differential. To reduce the number of necessary computations we used the same key, nonce, and counter for all output bits simultaneously. To test all combinations of rotation distances, we must execute Algorithm 1 2^{20} times. In addition, we defined the number of keys tested $N_k = 32$ and the number of tests per key $N_t = 1024$. Therefore, we have 2^{43} computation in total for 3 rounds of ChaCha. To achieve this amount of computation, we implemented Algorithm 1 in CUDA and executed it on a NVIDIA Quadro 4000 GPU, which required approximately 6 days of computation.

Algorithm 1 Differential Path Computation

procedure INPUT: A SET OF ROTATION DISTANCES r_1, r_2, r_3 AND r_4, THE NUMBER OF KEYS TESTED N_k, THE NUMBER OF TESTS PER KEY N_t
  $\varepsilon_d = 0$
  for each input differential $\Delta^0_{i,j}$ **do**
    $S = 0$
    for a from 1 to N_k **do**
      Generate random key k
      for b from 1 to N_t **do**
        Generate random nonce v
        Generate random counter t
        Initialize $X^{(0)}$ from k, v, t
        Compute $X^{(3)}$ from $X^{(0)}$
        Compute $X'^{(0)}$ from $X^{(0)}$ by flipping the bit $x^{(0)}_{i,j}$
        Compute $X'^{(3)}$ from $X'^{(0)}$
        $W = X^{(3)} \oplus X'^{(3)}$
        Convert W into a array of bits B
        $S = S + B$
    $m = \max(|2 \times S/(N_t N_k) - 1|)$
    if $m > \varepsilon_d$ **then**
      $\varepsilon_d = m$
  return ε_d

The results revealed that the bias of the differential path varies significantly for each combination of rotation distances. For example, if we set $r_1 = 0$ and $r_4 = 0$ (in other words, remove these rotations), we get the biases presented in Figure 1, which are

all equal, or very close to one. In comparison, if we set $r_1 = 16$ and $r_4 = 7$, we get much better results (see Figure 2) although there are still some very high biases for certain values of r_2 and r_3. Notice in Figure 2 that the maximum bias found for ChaCha with the original rotation distances $16, 12, 8, 7$ is not the best choice, since there are several combinations with smaller biases.

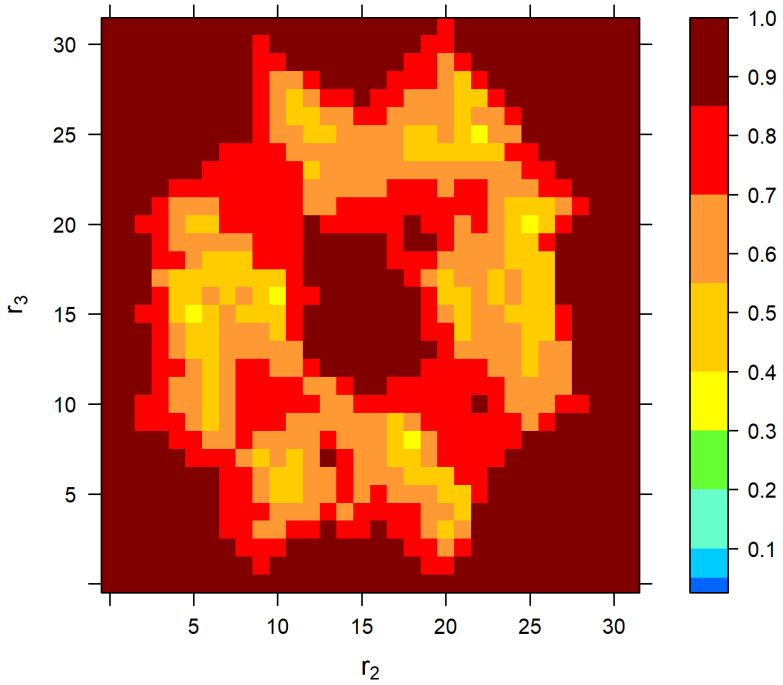


Figure 1. The biases were obtained for 3 rounds of ChaCha using rotations $r_1 = r_4 = 0$ and varying all values for r_2 and r_3. The color of the figure indicates the maximum absolute bias obtained for each combination of rotations. These are very poor results since that all biases are close to 1.

4.2. Finding Probabilistic Neutral Bits

From the results described in the previous section, we reduced the number of rotation distances under analysis by selecting the minimum bias available in the data and all the remaining biases that were statistically close to this minimum value. In total, remained 3162 combinations of rotation distances and the original set of rotation distances of ChaCha was not among these selected values. With the reduced set, we repeated the test of differential paths of the previous section but now with an increased value of $N_k = 256$, to achieve better precision.

The complexity of the attack depends not only on the bias of the differential path but also on the number of PNB. Thus, we performed another test to gather data about the behavior of PNB for each combination of rotation distances. It turns out that the computation necessary for this test increases significantly because we must test not only for each pair of input-output bits but also for each key bit individually. Fortunately, we empirically verified that, for ChaCha, the set of neutral bits are roughly the same for a particular output bit for any input bit. Thus, we can drastically reduce the necessary computation by randomly choosing the single-bit input differential. We computed the average neutrality for each output bit by performing 2^{16} iterations for each key bit, obtaining an array of 512

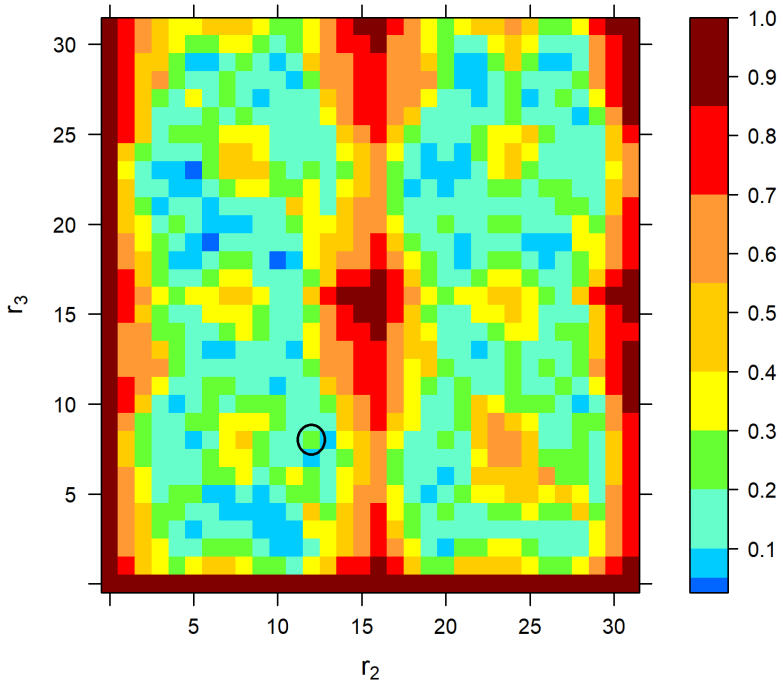


Figure 2. The biases we obtained for 3 rounds of ChaCha using rotations $r_1 = 16$ and $r_4 = 7$ and varying all values for r_2 and r_3. The color of the figure indicates the maximum absolute bias obtained for each combination of rotations. The value obtained for the original ChaCha is depicted inside a black circle.

values. Our final statistic is defined as the maximum value in this array. We performed this test considering 7 rounds of ChaCha.

After these tests, we chose our set rotation distances as $r_1 = 19$, $r_2 = 17$, $r_3 = 25$ and $r_4 = 11$, which are the values that minimize the product between both statistics. In particular, for this combination of rotation distances, we obtained 0.01497 for the bias of the differential path and 0.221 for the worst average neutrality. In the next section, we will show that this choice does improve the security of ChaCha against known attacks.

5. Security comparison

5.1. Estimating the Complexity of the PNB Attack

In [Aumasson et al. 2008], the authors reported an attack on 256-bit ChaCha20/6 and ChaCha20/7. For ChaCha20/6, they used the differential $(\Delta^3_{11,0}|\Delta^0_{13,13})$ with $|\varepsilon^\star_d| = 0.026$. The $\mathcal{OD}$ is observed after working 3 rounds backward from a 6-round keystream block. For the threshold $\gamma = 0.6$ they found a set of 147 non-significant key bits, with $|\varepsilon| = 0.00048$. This results in an attack in time 2^{139} and data 2^{30}. For ChaCha20/7, they used the same differential. The $\mathcal{OD}$ is observed after working 4 rounds backward from a 7-round keystream block. For the threshold $\gamma = 0.5$, they found a set of 35 non-significant key bits with $|\varepsilon| = 0.00059$. This results an attack in time 2^{248} and data 2^{27}.

We ran the attacks for ChaCha again, obtaining very similar complexity results. Using the same program, we ran the attack for ChaCha with rotation distances $19, 17, 25, 11$, showing that we get a stronger cipher. In fact, for 7 rounds, we did not find any attack with time $< 2^{256}$, see Table 2 for the results.

Algorithm	$\mathcal{ID}$	$\mathcal{OD}$	ε_d^*	γ	n	ϵ^*	Data	Time
ChaCha20/6	$\Delta^{(0)}_{12,21}$	$\Delta^{(3)}_{2,0}$	-0.1973	0.6	134	-0.0039	$2^{23.9}$	$2^{145.9}$
ChaCha20/7	$\Delta^{(0)}_{12,21}$	$\Delta^{(3)}_{2,0}$	-0.1977	0.4	20	-0.0097	$2^{17.8}$	2^{254}
*ChaCha20/6	$\Delta^{(0)}_{14,17}$	$\Delta^{(3)}_{1,0}$	-0.0059	0.8	111	-0.0019	$2^{25.6}$	$2^{170.6}$
*ChaCha20/7	–	–	–	–	–	–	–	–

Table 2. Best attacks obtained for ChaCha and for its modified version with rotation distances $19, 17, 25, 11$, denoted here by *ChaCha. We could not find any attacks for the modified version of ChaCha with 7 rounds.

5.2. Multi-bit differential

In [Choudhuri and Maitra 2016], the authors provides several different attacks for ChaCha20/4, ChaCha20/5, ChaCha20/6, and ChaCha20/7. For ChaCha20/4, Lemma 1 is used. Considering the first row of Table 3, we note a bias $\varepsilon_d = 0.1984$ and thus $\frac{1}{\varepsilon_d^2/2} < 51$. That is, with 2^6 samples it is enough to distinguish 4-round ChaCha from a uniform random source. However, when changing the rotation distances, the best bias we get is $\varepsilon_d = -0.009179$ and thus $\frac{1}{\varepsilon_d^2/2} < 23738$. That is, with 2^{15} samples it is enough to distinguish 4-round ChaCha with rotation distances $19, 17, 25, 11$ from a uniform random source.

For ChaCha20/5, if we define $\mathcal{ID}$ at $\Delta x^0_{13,13}$ and $\mathcal{OD}$ at $\Delta x^3_{11,0}$, we obtain $\varepsilon_d = -0.0272$. By Lemma 1, we can extend this bias to 4 rounds, and by Lemma 2, we can further extend this bias to 5 rounds with probability $3/4$, or $\varepsilon_L = 1/2$. This gives a total differential-linear 5-th round bias of $\varepsilon = \varepsilon_d \varepsilon_L^2 = -0.0068$ thus $\frac{1}{\varepsilon^2/2} < 43253$. That is, with 2^{16} samples it is enough to distinguish 5-round ChaCha from a uniform random source. However, changing the rotation distances and if we define $\mathcal{ID}$ at $\Delta^0_{14,12}$ and $\mathcal{OD}$ at $\Delta^3_{8,0}$, we obtain $\varepsilon_d = -0.000915$, and from Lemmas 1 and 2, we get a total differential-linear 5-th round bias of $\varepsilon = \varepsilon_d \varepsilon_L^2 = -0.00022875$ thus $\frac{1}{\varepsilon^2/2} < 38221506$. That is, with 2^{26} samples it is enough to distinguish 5-round ChaCha with rotation distances $19, 17, 25, 11$ from a uniform random source.

Algorithm	$\mathcal{ID}$	$\mathcal{OD}$	Bias
ChaCha	$\Delta x^{(0)}_{12,20}$	$\Delta x^{(4)}_{2,0} \oplus \Delta x^{(4)}_{7,7} \oplus \Delta x^{(4)}_{7,19} \oplus \Delta x^{(4)}_{8,12} \oplus \Delta x^{(4)}_{13,0}$	0.1984
ChaCha	$\Delta x^{(0)}_{14,20}$	$\Delta x^{(4)}_{0,0} \oplus \Delta x^{(4)}_{5,7} \oplus \Delta x^{(4)}_{5,19} \oplus \Delta x^{(4)}_{10,12} \oplus \Delta x^{(4)}_{15,0}$	0.1979
ChaCha	$\Delta x^{(0)}_{15,20}$	$\Delta x^{(4)}_{1,0} \oplus \Delta x^{(4)}_{6,7} \oplus \Delta x^{(4)}_{6,19} \oplus \Delta x^{(4)}_{11,12} \oplus \Delta x^{(4)}_{12,0}$	0.1973
ChaCha	$\Delta x^{(0)}_{13,20}$	$\Delta x^{(4)}_{3,0} \oplus \Delta x^{(4)}_{4,7} \oplus \Delta x^{(4)}_{4,19} \oplus \Delta x^{(4)}_{9,12} \oplus \Delta x^{(4)}_{14,0}$	0.1972
*ChaCha	$\Delta x^{(0)}_{14,1}$	$\Delta x^{(4)}_{0,0} \oplus \Delta x^{(4)}_{5,11} \oplus \Delta x^{(4)}_{5,28} \oplus \Delta x^{(4)}_{10,17} \oplus \Delta x^{(4)}_{15,0}$	-0.009179
*ChaCha	$\Delta x^{(0)}_{15,16}$	$\Delta x^{(4)}_{0,0} \oplus \Delta x^{(4)}_{5,11} \oplus \Delta x^{(4)}_{5,28} \oplus \Delta x^{(4)}_{10,17} \oplus \Delta x^{(4)}_{15,0}$	-0.009133
*ChaCha	$\Delta x^{(0)}_{15,1}$	$\Delta x^{(4)}_{1,0} \oplus \Delta x^{(4)}_{6,11} \oplus \Delta x^{(4)}_{6,28} \oplus \Delta x^{(4)}_{11,17} \oplus \Delta x^{(4)}_{12,0}$	-0.009122
*ChaCha	$\Delta x^{(0)}_{14,16}$	$\Delta x^{(4)}_{3,0} \oplus \Delta x^{(4)}_{4,11} \oplus \Delta x^{(4)}_{4,28} \oplus \Delta x^{(4)}_{9,17} \oplus \Delta x^{(4)}_{14,0}$	-0.009099

Table 3. The best multi-bit differentials for ChaCha and for its modified version with rotation distances $19, 17, 25, 11$, denoted here by *ChaCha. Notice that we can reduce the bias significantly.

Extending the linear approximation for 3 rounds comes at a cost. As discussed in

[Choudhuri and Maitra 2016], for 6 rounds, the linear bias after expanding any equation from Lemma 2 is $\varepsilon_L = 1/(2 \cdot 1 + 3 \cdot 4 + 5 \cdot 1 + 3 \cdot 2 + 2 \cdot 1) = 1/2^{26}$. To use this extension, we searched for the input differential which maximizes the differential bias for $\Delta x^{(3)}_{8,0}, \Delta x^{(3)}_{9,0}, \Delta x^{(3)}_{10,0}$ or $\Delta x^{(3)}_{11,0}$, which leads to the differential pair $(\Delta x^{(3)}_{9,0} | \Delta x^{(0)}_{15,12})$ with $\varepsilon_d = 0.000792$. This leads to a 6-round bias of $\varepsilon_L^2 \varepsilon_d \approx \frac{1}{2^{62.3}}$ and a distinguisher with complexity of 2^{125}.

For a key recovery attack against 6 rounds of ChaCha, we must use PNB. The best attack we obtained when considering the proposed rotation distances uses 5 rounds forward and then 1 round backward. For this the $\mathcal{ID}$ is $\Delta^{(0)}_{12,12}$ and the $\mathcal{OD}$ in the third round is $\Delta^{(3)}_{10,0}$, thus, using the third equation of Lemma 2, we can mount an attack. We got 157 PNBs using $\gamma = 0.6$ from which we estimated $\varepsilon = -0.000024$, $\varepsilon^\star = -0.000023$ leading to an attack with data complexity $2^{38.7}$ and time complexity $2^{137.7}$. For 7 rounds of ChaCha with the proposed rotation distances, we did not find any significant attacks. Also, we could not use the equations from Lemma 4 since we could not find any significant bias for a double output differential bias. Table 4 summarizes our findings.

Algorithm	Rounds	Data	Time	Type	Reference
ChaCha	4	2^6	2^6	Distinguisher	[Choudhuri and Maitra 2016]
	5	2^{16}	2^{16}	Distinguisher	[Choudhuri and Maitra 2016]
	6	2^{75}	2^{75}	Distinguisher	[Coutinho and Neto 2020]
	6	2^{56}	$2^{102.2}$	Key recovery	[Coutinho and Neto 2020]
	7	2^{50}	$2^{231.9}$	Key recovery	[Coutinho and Neto 2020]
*ChaCha	4	2^{15}	2^{15}	Distinguisher	This work
	5	2^{26}	2^{26}	Distinguisher	This work
	6	2^{125}	2^{125}	Distinguisher	This work
	6	$2^{38.7}$	$2^{137.7}$	Key recovery	This work
	7	–	–	–	–

Table 4. Attacks obtained considering the techniques presented in Section 3.2. Notice that the complexity of the attacks for ChaCha with rotation distances $19, 17, 25, 11$, denoted here by *ChaCha, are higher, thus, the proposed modification is more secure against these attacks.

6. Conclusion

In this work, we proposed a modification for the stream cipher ChaCha. This was done by considering the best attacks in the literature and trying to minimize the differential biases generated by 3 rounds of the algorithm. This analysis resulted in the optimal rotation distances for ChaCha against differential-linear cryptanalysis, which are $r_1 = 19$, $r_2 = 17$, $r3 = 25$, and $r_4 = 11$. We computed the complexity of the two most successful attacks against ChaCha presented in the literature, showing that the proposed modification leads to attacks with higher complexity for 4, 5, and 6 rounds (see Table 4). For 7 rounds, ChaCha with the proposed rotation distances is no longer vulnerable to differential attacks. For future work, it remains to test other types of attacks known in the literature, in particular, related key attacks and the attack of Beierle et al. [Beierle et al. 2020], published after the submission of this work.

Acknowledgements

This work was supported in part by CNPq - Brazilian National Research Council (Grants 312180/2019-5 PQ-2, BRICS2017-591 LargEWiN, and 465741/2014-2 INCT on Cybersecurity), in part by CAPES - Brazilian Higher Education Personnel Improvement Coordination (Grants PROAP PPGEE/UnB, 23038.007604/2014-69 FORTE, and 88887.144009/2017-00 PROBRAL), in part by FAP-DF - Brazilian Federal District Research Support Foundation (Grant 0193.001366/2016 UIoT, and Grant 0193.001365/2016 SSDDC), in part by the Brazilian Ministry of the Economy (Grant 005/2016 DIPLA, and Grant 083/2016 ENAP), in part by the Institutional Security Office of the Presidency of Brazil (Grant ABIN 002/2017), in part by the Administrative Council for Economic Defense (Grant CADE 08700.000047/2019-14), and in part by the General Attorney of the Union (Grant AGU 697.935/2019).

References

[Aumasson et al. 2008] Aumasson, J.-P., Fischer, S., Khazaei, S., Meier, W., and Rechberger, C. (2008). New features of latin dances: analysis of Salsa, ChaCha, and Rumba. In *International Workshop on Fast Software Encryption*, pages 470–488. Springer.

[Beierle et al. 2020] Beierle, C., Leander, G., and Todo, Y. (2020). Improved differential-linear attacks with applications to ARX ciphers. In *Annual International Cryptology Conference*, pages 329–358. Springer.

[Bernstein 2005] Bernstein, D. J. (2005). The Poly1305-AES message-authentication code. In *International Workshop on Fast Software Encryption*, pages 32–49. Springer.

[Bernstein 2008a] Bernstein, D. J. (2008a). ChaCha, a variant of Salsa20. In *Workshop Record of SASC*, volume 8, pages 3–5.

[Bernstein 2008b] Bernstein, D. J. (2008b). The Salsa20 family of stream ciphers. In *New stream cipher designs*, pages 84–97. Springer.

[Choudhuri and Maitra 2016] Choudhuri, A. R. and Maitra, S. (2016). Significantly improved multi-bit differentials for reduced round Salsa and Chacha. *IACR Transactions on Symmetric Cryptology*, pages 261–287.

[Coutinho et al. 2020] Coutinho, M., De Sousa, R. T., and Borges, F. (2020). Continuous diffusion analysis. *IEEE Access*.

[Coutinho and Neto 2020] Coutinho, M. and Neto, T. C. S. (2020). New multi-bit differentials to improve attacks against ChaCha. Cryptology ePrint Archive, Report 2020/350. `https://eprint.iacr.org/2020/350`.

[Crowley 2006] Crowley, P. (2006). Truncated differential cryptanalysis of five rounds of Salsa20. *The State of the Art of Stream Ciphers SASC*, 2006:198–202.

[Dey et al. 2019] Dey, S., Roy, T., and Sarkar, S. (2019). Revisiting design principles of Salsa and ChaCha. *Advances in Mathematics of Communications*, 13(4).

[Dey and Sarkar 2017] Dey, S. and Sarkar, S. (2017). Improved analysis for reduced round Salsa and ChaCha. *Discrete Applied Mathematics*, 227:58–69.

[Ding 2019] Ding, L. (2019). Improved related-cipher attack on Salsa20 stream cipher. *IEEE Access*, 7:30197–30202.

[Fischer et al. 2006] Fischer, S., Meier, W., Berbain, C., Biasse, J.-F., and Robshaw, M. J. (2006). Non-randomness in eSTREAM candidates Salsa20 and TSC-4. In *International Conference on Cryptology in India*, pages 2–16. Springer.

[Hernandez-Castro et al. 2008] Hernandez-Castro, J. C., Tapiador, J. M., and Quisquater, J.-J. (2008). On the Salsa20 core function. In *International Workshop on Fast Software Encryption*, pages 462–469. Springer.

[IANIX 2020] IANIX (2020). ChaCha usage & deployment. `https://ianix.com/pub/chacha-deployment.html`. Accessed: 2020-01-13.

[Ishiguro et al. 2011] Ishiguro, T., Kiyomoto, S., and Miyake, Y. (2011). Latin dances revisited: new analytic results of Salsa20 and ChaCha. In *International Conference on Information and Communications Security*, pages 255–266. Springer.

[Langley et al. 2016] Langley, A., Chang, W., Mavrogiannopoulos, N., Strombergson, J., and Josefsson, S. (2016). ChaCha20-Poly1305 cipher suites for transport layer security (TLS). *RFC 7905*, (10).

[Maitra 2016] Maitra, S. (2016). Chosen IV cryptanalysis on reduced round ChaCha and Salsa. *Discrete Applied Mathematics*, 208:88–97.

[Maitra et al. 2015] Maitra, S., Paul, G., and Meier, W. (2015). Salsa20 cryptanalysis: New moves and revisiting old styles. In *the Ninth International Workshop on Coding and Cryptography*.

[Mouha and Preneel 2013] Mouha, N. and Preneel, B. (2013). A proof that the ARX cipher Salsa20 is secure against differential cryptanalysis. *IACR Cryptology ePrint Archive*, 2013:328.

[Shi et al. 2012] Shi, Z., Zhang, B., Feng, D., and Wu, W. (2012). Improved key recovery attacks on reduced-round Salsa20 and ChaCha. In *International Conference on Information Security and Cryptology*, pages 337–351. Springer.

[Torvalds 2016] Torvalds, L. (2016). Linux kernel source tree. `https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=818e607b57c94ade9824dad63a96c2ea6b21baf3`.

[Tsunoo et al. 2007] Tsunoo, Y., Saito, T., Kubo, H., Suzaki, T., and Nakashima, H. (2007). Differential cryptanalysis of Salsa20/8. In *Workshop Record of SASC*, volume 28.

Artigos publicados em 2021



Improved Linear Approximations to ARX Ciphers and Attacks Against ChaCha

Murilo Coutinho( ) and Tertuliano C. Souza Neto

Research and Development Center for Communications Security (CEPESC), Rio de Janeiro, Brazil
murilo.coutinho@redes.unb.br

Abstract. In this paper, we present a new technique which can be used to find better linear approximations in ARX ciphers. Using this technique, we present the first explicitly derived linear approximations for 3 and 4 rounds of ChaCha and, as a consequence, it enables us to improve the recent attacks against ChaCha. Additionally, we present new differentials for 3 and 3.5 rounds of ChaCha that, when combined with the proposed technique, lead to further improvement in the complexity of the Differential-Linear attacks against ChaCha.

Keywords: Differential-linear cryptanalysis · ARX-Ciphers · ChaCha

1 Introduction

Symmetric cryptographic primitives are heavily used in a variety of contexts. In particular, ARX-based design is a major building block of modern ciphers due to its efficiency in software. ARX stands for addition, word-wise rotation and XOR. Indeed, ciphers following this framework are composed of those operations and avoid the computation of smaller S-boxes through look-up tables. The ARX-based design approach is used to design stream ciphers (e.g., Salsa20 [7] and ChaCha [6]), efficient block ciphers (e.g., Sparx [15]), cryptographic permutations (e.g., Sparkle [3]) and hash functions (e.g., Blake [2]).

ARX-based designs are not only efficient but provide good security properties. The algebraic degree of ARX ciphers is usually high after only a very few rounds as the carry bit within one modular addition already reaches almost maximal degree. For differential and linear attacks, ARX-based designs show weaknesses for a small number of rounds. However, after some rounds the differential and linear probabilities decrease rapidly. Thus, the probabilities of differentials and the absolute correlations of linear approximations decrease very quickly as we increase the number of rounds. In fact, this property led to the long-trail strategy for designing ARX-based ciphers [15].

Ciphers and primitives based on Salsa20 and ChaCha families are heavily used in practice. In 2005, Bernstein proposed the stream cipher Salsa20 [7] as a contender to the eSTREAM [26], the ECRYPT Stream Cipher Project. As

© International Association for Cryptologic Research 2021
A. Canteaut and F.-X. Standaert (Eds.): EUROCRYPT 2021, LNCS 12696, pp. 711–740, 2021.
https://doi.org/10.1007/978-3-030-77870-5_25

outlined by the author, Salsa20 is an ARX type family of algorithms which can be ran with several number of rounds, including the well known Salsa20/12 and Salsa20/8 versions. Latter, in 2008, Bernstein proposed some modifications to Salsa20 in order to provide better diffusion per round and higher resistance to cryptanalysis. These changes originated a new stream cipher, a variant which he called ChaCha [6]. Although Salsa20 was one of the winners of the eSTREAM competition, ChaCha has received much more attention through the years. Nowadays, we see the usage of this cipher in several projects and applications.

ChaCha, along with Poly1305 [5], is in one of the cipher suits of the new TLS 1.3 [21], which has been used by Google on both Chrome and Android. Not only has ChaCha been used in TLS but also in many other protocols such as SSH, Noise and S/MIME 4.0. In addition, the RFC 7634 proposes the use of ChaCha in IKE and IPsec. ChaCha has been used not only for encryption, but also as a pseudo-random number generator in any operating system running Linux kernel 4.8 or newer [25,28]. Additionally, ChaCha has been used in several applications such as WireGuard (VPN) (see [18] for a huge list of applications, protocols and libraries using ChaCha).

Related Work. Since ChaCha is so heavily used, it is very important to understand its security. Indeed, the cryptanalysis of ChaCha is well understood and several authors studied its security [1,9,11–14,16,17,19,22–24,27,29] which show weaknesses in the reduced round versions of the cipher.

The cryptanalysis of Salsa20 was introduced by Crowley [11] in 2005. Crowley developed a differential attack against Salsa20/5, namely the 5-round version of Salsa20, and received the $1000 prize offered by Bernstein for the most interesting Salsa20 cryptanalysis in that year. In 2006, Fischer et al. [16] improved the attack against Salsa20/5 and presented their attack against Salsa20/6.

Probably the most important cryptanalysis in this regard was proposed by Aumasson et al. at FSE 2008 [1] with the introduction of Probabilistic Neutral Bits (PNBs), showing attacks against Salsa20/7, Salsa20/8, ChaCha20/6 and ChaCha20/7. After that, several authors proposed small enhancements on the attack of Aumasson et al. The work by Shi et al. [27] introduced the concept of Column Chaining Distinguisher (CCD) to achieve some incremental advancements over [1] for both Salsa and ChaCha.

Maitra, Paul and Meier [22] studied an interesting observation regarding round reversal of Salsa, but no significant cryptanalytic improvement could be obtained using this method. Maitra [23] used a technique of Chosen IVs to obtain certain improvements over existing results. Dey and Sarkar [13] showed how to choose values for the PNB to further improve the attack.

In a paper presented in FSE 2017, Choudhuri and Maitra [9] significantly improved the attacks by considering the mathematical structure of both Salsa and ChaCha in order to find differential characteristics with much higher correlations. Recently, Coutinho and Souza [10] proposed new multi-bit differentials using the mathematical framework of Choudhuri and Maitra. In Crypto 2020, Beierle et al. [4] proposed improvements to the framework of differential-linear cryptanalysis against ARX-based designs and further improved the attacks against ChaCha.

Our Contribution. In this work, we provide a new framework to find linear approximations for ARX ciphers. Using this framework we provide the first explicitly derived linear approximations for 3 and 4 rounds of ChaCha. Exploring these linear approximations, we can improve the attacks for 6 and 7 rounds of ChaCha. Additionally, we present new differentials for 3 and 3.5 rounds of ChaCha which also improve the attacks. We summarize our findings along with other significant attacks for comparison in Table 1. Also, we verified all theoretical results with random experiments. We provide the source code to reproduce this paper in Github https://github.com/MurCoutinho/cryptanalysisChaCha.git, which is, for the best of our knowledge, the first implementation of cryptanalysis against ChaCha available to the public. We should note that it is possible to find attacks with less complexity for related key attacks, but we do not consider them in this work.

Table 1. The best attacks against ChaCha with 256-bit key.

Rounds	Time Complexity	Data Complexity	Reference
4	2^{6}	2^{6}	[9]
4.5	2^{12}	2^{12}	[9]
5	2^{16}	2^{16}	[9]
6	2^{139}	2^{30}	[1]
	2^{136}	2^{28}	[27]
	2^{130}	2^{35}	[9]
	$2^{127.5}$	$2^{37.5}$	[9]
	2^{116}	2^{116}	[9]
	$2^{102.2}$	2^{56}	[10]
	$2^{77.4}$	2^{58}	[4]
	2^{75}	2^{75}	[10]
	2^{51}	2^{51}	This work
7	2^{248}	2^{27}	[1]
	$2^{246.5}$	2^{27}	[27]
	$2^{238.9}$	2^{96}	[23]
	$2^{237.7}$	2^{96}	[9]
	$2^{231.9}$	2^{50}	[10]
	$2^{230.86}$	$2^{48.8}$	[4]
	$2^{228.51}$	$2^{80.51}$	This work
	2^{224}	2^{224}	This work
	2^{218}	2^{218}	This work

Organization of the Paper. In Sect. 2, we provide an overview of previous results, including a description of ChaCha, a summary of differential-linear cryptanalysis and a review of the techniques developed by Choudhuri and Maitra in

[9]. In Sect. 3, we present a new technique which can be used to find better linear approximations in ARX ciphers and theoretically develop new linear relations between bits of different rounds for ChaCha. Then, in Sect. 4, we show that these new linear approximations lead to a better distinguisher and key recovery attacks of ChaCha reduced to 6 and 7 rounds. Finally, Sect. 5 presents the conclusion and future work.

2 Specifications and Preliminaries

The main notation we will use throughout the paper is defined in Table 2. Next we define the algorithm ChaCha.

Table 2. Notation

Notation	Description
X	a 4×4 state matrix of ChaCha
$X^{(0)}$	initial state matrix of ChaCha
$X^{(R)}$	state matrix after application of R round functions
Z	output of ChaCha, $Z = X^{(0)} + X^{(R)}$
$x_i^{(R)}$	i^{th} word of the state matrix $X^{(R)}$ (words arranged in row major)
$x_{i,j}^{(R)}$	j^{th} bit of ith word of the state matrix $X^{(R)}$
$x_i^{(R)}[j_0, j_1, ..., j_t]$	the sum $x_{i,j_0}^{(R)} \oplus x_{i,j_1}^{(R)} \oplus \cdots \oplus x_{i,j_t}^{(R)}$
$x + y$	addition of x and y modulo 2^{32}
$x - y$	subtraction of x and y modulo 2^{32}
$x \oplus y$	bitwise XOR of x and y
$x \lll n$	rotation of x by n bits to the left
$x \ggg n$	rotation of x by n bits to the right
Δx	XOR difference of x and x'. $\Delta x = x \oplus x'$
$\Delta X^{(R)}$	XOR difference of $X^{(R)}$ and $X'^{(R)}$. $\Delta X^{(R)} = X^{(R)} \oplus X'^{(R)}$
$\Delta x_i^{(R)}$	differential $\Delta x_i^{(R)} = x_i^{(R)} \oplus x'^{(R)}_i$
$\Delta x_{i,j}^{(R)}$	differential $\Delta x_{i,j}^{(R)} = x_{i,j}^{(R)} \oplus x'^{(R)}_{i,j}$
$\Pr(E)$	probability of occurrence of an event E
$\mathcal{I}D$	input difference
$\mathcal{O}D$	output difference

2.1 ChaCha

The stream cipher Salsa20 was proposed by Bernstein [7] to the *eSTREAM* competition and later Bernstein proposed ChaCha [6] as an improvement of Salsa20. ChaCha consists of a series of ARX (addition, rotation, and XOR) operations on 32-bit words, being highly efficient in software and hardware.

Each round of ChaCha has a total of 16 bitwise XOR, 16 addition modulo 2^{32} and 16 constant-distance rotations.

ChaCha operates on a state of 64 bytes, organized as a 4×4 matrix with 32-bit integers, initialized with a 256-bit key $k_0, k_1, ..., k_7$, a 64-bit nonce v_0, v_1 and a 64-bit counter t_0, t_1 (we may also refer to the nonce and counter words as IV words), and 4 constants $c_0 = 0\text{x}61707865$, $c_1 = 0\text{x}3320646e$, $c_2 = 0\text{x}79622d32$ and $c_3 = 0\text{x}6b206574$. For ChaCha, we have the following initial state matrix:

$$X^{(0)} = \begin{pmatrix} x_0^{(0)} & x_1^{(0)} & x_2^{(0)} & x_3^{(0)} \\ x_4^{(0)} & x_5^{(0)} & x_6^{(0)} & x_7^{(0)} \\ x_8^{(0)} & x_9^{(0)} & x_{10}^{(0)} & x_{11}^{(0)} \\ x_{12}^{(0)} & x_{13}^{(0)} & x_{14}^{(0)} & x_{15}^{(0)} \end{pmatrix} = \begin{pmatrix} c_0 & c_1 & c_2 & c_3 \\ k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ t_0 & t_1 & v_0 & v_1 \end{pmatrix}. \tag{1}$$

The state matrix is modified in each round by a *Quarter Round Function* (QRF), denoted by $QR\left(x_a^{(r-1)}, x_b^{(r-1)}, x_c^{(r-1)}, x_d^{(r-1)}\right)$, which receives and updates 4 integers in the following way:

$$\begin{array}{llll} x_{a\prime}^{(r-1)} = x_a^{(r-1)} + x_b^{(r-1)}; & x_{d\prime}^{(r-1)} = (x_d^{(r-1)} \oplus x_{a\prime}^{(r-1)}) \lll 16; \\ x_{c\prime}^{(r-1)} = x_c^{(r-1)} + x_{d\prime}^{(r-1)}; & x_{b\prime}^{(r-1)} = (x_b^{(r-1)} \oplus x_{c\prime}^{(r-1)}) \lll 12; \\ x_a^{(r)} = x_{a\prime}^{(r-1)} + x_{b\prime}^{(r-1)}; & x_d^{(r)} = (x_{d\prime}^{(r-1)} \oplus x_a^{(r)}) \lll 8; \\ x_c^{(r)} = x_{c\prime}^{(r-1)} + x_d^{(r)}; & x_b^{(r)} = (x_{b\prime}^{(r-1)} \oplus x_c^{(r)}) \lll 7; \end{array} \tag{2}$$

One round of ChaCha is defined as 4 applications of the QRF. There is, however, a difference between odd and even rounds. For odd rounds, i.e. $r \in \{1, 3, 5, 7, ...\}$, $X^{(r)}$ is obtained from $X^{(r-1)}$ by applying

$$\begin{array}{l} \left(x_0^{(r)}, x_4^{(r)}, x_8^{(r)}, x_{12}^{(r)}\right) = QR\left(x_0^{(r-1)}, x_4^{(r-1)}, x_8^{(r-1)}, x_{12}^{(r-1)}\right) \\ \left(x_1^{(r)}, x_5^{(r)}, x_9^{(r)}, x_{13}^{(r)}\right) = QR\left(x_1^{(r-1)}, x_5^{(r-1)}, x_9^{(r-1)}, x_{13}^{(r-1)}\right) \\ \left(x_2^{(r)}, x_6^{(r)}, x_{10}^{(r)}, x_{14}^{(r)}\right) = QR\left(x_2^{(r-1)}, x_6^{(r-1)}, x_{10}^{(r-1)}, x_{14}^{(r-1)}\right) \\ \left(x_3^{(r)}, x_7^{(r)}, x_{11}^{(r)}, x_{15}^{(r)}\right) = QR\left(x_3^{(r-1)}, x_7^{(r-1)}, x_{11}^{(r-1)}, x_{15}^{(r-1)}\right) \end{array}.$$

On the other hand, for even rounds, i.e. $r \in \{2, 4, 6, 8, , ...\}$, $X^{(r)}$ is calculated from $X^{(r-1)}$ by applying

$$\begin{array}{l} \left(x_0^{(r)}, x_5^{(r)}, x_{10}^{(r)}, x_{15}^{(r)}\right) = QR\left(x_0^{(r-1)}, x_5^{(r-1)}, x_{10}^{(r-1)}, x_{15}^{(r-1)}\right) \\ \left(x_1^{(r)}, x_6^{(r)}, x_{11}^{(r)}, x_{12}^{(r)}\right) = QR\left(x_1^{(r-1)}, x_6^{(r-1)}, x_{11}^{(r-1)}, x_{12}^{(r-1)}\right) \\ \left(x_2^{(r)}, x_7^{(r)}, x_8^{(r)}, x_{13}^{(r)}\right) = QR\left(x_2^{(r-1)}, x_7^{(r-1)}, x_8^{(r-1)}, x_{13}^{(r-1)}\right) \\ \left(x_3^{(r)}, x_4^{(r)}, x_9^{(r)}, x_{14}^{(r)}\right) = QR\left(x_3^{(r-1)}, x_4^{(r-1)}, x_9^{(r-1)}, x_{14}^{(r-1)}\right) \end{array}.$$

The output of ChaCha20/R is then defined as the sum of the initial state with the state after R rounds $Z = X^{(0)} + X^{(R)}$. One should note that it is possible to parallelize each application of the QRF on each round and also that each round is reversible. Hence, we can compute $X^{(r-1)}$ from $X^{(r)}$. For more information on ChaCha, we refer to [6].

2.2 Differential-Linear Cryptanalysis

In this section, we describe the technique of Differential-Linear cryptanalysis as used to attack ChaCha. Let E be a cipher and suppose we can write $E = E_2 \circ E_1$, where E_1 and E_2 are sub ciphers, covering m and l rounds of the main cipher, respectively. We can apply an input difference $\mathcal{ID}$ $\Delta X^{(0)}$ in the sub cipher E_1 obtaining an output difference $\mathcal{OD}$ $\Delta X^{(m)}$ (see the left side of Fig. 1). The next step is to apply Linear Cryptanalysis to the second sub cipher E_2. Using masks Γ_m and Γ_{out}, we attempt to find good linear approximations covering the remaining l rounds of the cipher E. Applying this technique we can construct a differential-linear distinguisher covering all $m + l$ rounds of the cipher E. This is the main idea in Langford and Hellman's classical approach [20].

Sometimes, however, it can be useful to divide the cipher E into three other ciphers, i.e. $E = E_3 \circ E_2 \circ E_1$. In this scenario, we can explore properties of the cipher in the first part E_1, and then apply a differential linear attack where we divide the differential part of the attack in two (see the right side of Fig. 1). Here, the $\mathcal{OD}$ from the sub cipher E_1 after r rounds, namely $\Delta X^{(r)}$, is the $\mathcal{ID}$ for the sub cipher E_2 which produces an output difference $\Delta X^{(m)}$. For more information in this regard, see [4].

It is important to understand how to compute the complexity of a differential-linear attack. We denote the differential of the state matrix as $\Delta X^{(r)} = X^{(r)} \oplus X'^{(r)}$ and the differential of individual words as $\Delta x_i^{(r)} = x_i^{(r)} \oplus x_i'^{(r)}$. Let $x_{i,j}^{(r)}$ denote the j-th bit of the i-th word of the state matrix after r rounds and let $\mathcal{J}$ be a set of bits. Also, let σ and σ' be linear combinations of bits in the set $\mathcal{J}$

$$\sigma = \left(\bigoplus_{(i,j)\in\mathcal{J}} x_{i,j}^{(r)} \right), \qquad \sigma' = \left(\bigoplus_{(i,j)\in\mathcal{J}} x_{i,j}'^{(r)} \right).$$

Then

$$\Delta\sigma = \left(\bigoplus_{(i,j)\in\mathcal{J}} \Delta x_{i,j}^{(r)} \right)$$

is the linear combination of the differentials. We can write

$$\Pr\left[\Delta\sigma = 0 | \Delta X^{(0)}\right] = \frac{1}{2}(1 + \varepsilon_d), \tag{3}$$

where ε_d is the differential correlation.

Using linear cryptanalysis, it is possible to go further and find new relations between the initial state matrix and the state matrix after $R > r$ rounds. To do so, let $\mathcal{L}$ denote another set of bits and define

$$\rho = \left(\bigoplus_{(i,j)\in\mathcal{L}} x_{i,j}^{(R)} \right), \qquad \rho' = \left(\bigoplus_{(i,j)\in\mathcal{L}} x_{i,j}'^{(R)} \right).$$

Then, as before,

$$\Delta\rho = \left(\bigoplus_{(i,j)\in\mathcal{L}} \Delta x_{i,j}^{(R)} \right).$$

We can define $\Pr[\sigma = \rho] = \frac{1}{2}(1 + \varepsilon_L)$, where ε_L is the linear correlation. We want to find γ such that $\Pr\left[\Delta\rho = 0 | \Delta X^{(0)}\right] = \frac{1}{2}(1 + \gamma)$.

To compute γ, we write (to simplify the notation we make the conditional to $\Delta X^{(0)}$ implicit):

$$\begin{aligned}\Pr[\Delta\sigma = \Delta\rho] &= \Pr[\sigma = \rho] \cdot \Pr\left[\sigma' = \rho'\right] + \Pr[\sigma = \bar{\rho}] \cdot \Pr\left[\sigma' = \overline{\rho'}\right] \\ &= \frac{1}{2}\left(1 + \varepsilon_L^2\right).\end{aligned}$$

Then,

$$\begin{aligned}\Pr[\Delta\rho = 0] &= \Pr[\Delta\sigma = 0] \cdot \Pr[\Delta\sigma = \Delta\rho] + \Pr[\Delta\sigma = 1] \cdot \Pr[\Delta\sigma = \overline{\Delta\rho}] \\ &= \frac{1}{2}\left(1 + \varepsilon_d \cdot \varepsilon_L^2\right).\end{aligned}$$

Therefore, the differential-linear correlation is given by $\gamma = \varepsilon_d \cdot \varepsilon_L^2$, which defines a distinguisher with complexity $\mathcal{O}\left(\frac{1}{\varepsilon_d^2 \varepsilon_L^4}\right)$. For further information on differential-linear cryptanalysis we refer to [8].

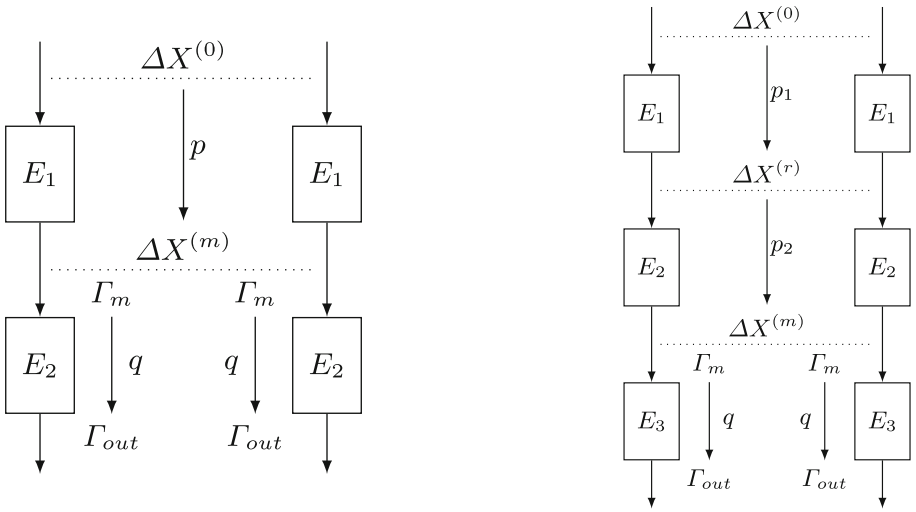


Fig. 1. A classical differential-linear distinguisher (on the left) and a differential-linear distinguisher with experimental evaluation of the correlation p_2 (on the right).

2.3 Multi-bit Differential for Reduced Round ChaCha

In this section, we review the work presented in [9] and in [10]. In these works, the authors developed the theory for selecting specific combination of bits to give

high correlations for Chacha. To do that, in both papers the authors analyzed the QRF directly, representing each equation in its bit level. In the following, we change the original notation of the referred papers in order to create a notation that will be better for the purposes of this work.

Thus, let $\Theta(x, y) = x \oplus y \oplus (x + y)$ be the carry function of the sum $x + y$. Define $\Theta_i(x, y)$ as the i-th bit of $\Theta(x, y)$. By definition, we have $\Theta_0(x, y) = 0$. We can write the QRF equations of ChaCha (Eq. 2) as

$$\begin{aligned}
x'^{(m-1)}_{a,i} &= x^{(m-1)}_{a,i} \oplus x^{(m-1)}_{b,i} \oplus \Theta_i(x^{(m-1)}_a, x^{(m-1)}_b)\\
x'^{(m-1)}_{d,i+16} &= x^{(m-1)}_{d,i} \oplus x'^{(m-1)}_{a,i}\\
x'^{(m-1)}_{c,i} &= x^{(m-1)}_{c,i} \oplus x'^{(m-1)}_{d,i} \oplus \Theta_i(x^{(m-1)}_c, x'^{(m-1)}_d)\\
x'^{(m-1)}_{b,i+12} &= x^{(m-1)}_{b,i} \oplus x'^{(m-1)}_{c,i}\\
x^{(m)}_{a,i} &= x'^{(m-1)}_{a,i} \oplus x'^{(m-1)}_{b,i} \oplus \Theta_i(x'^{(m-1)}_a, x'^{(m-1)}_b)\\
x^{(m)}_{d,i+8} &= x'^{(m-1)}_{d,i} \oplus x^{(m)}_{a,i}\\
x^{(m)}_{c,i} &= x'^{(m-1)}_{c,i} \oplus x^{(m)}_{d,i} \oplus \Theta_i(x'^{(m-1)}_c, x^{(m)}_d)\\
x^{(m)}_{b,i+7} &= x'^{(m-1)}_{b,i} \oplus x^{(m)}_{c,i}
\end{aligned} \tag{4}$$

Inverting these equations, we get:

$$x'^{(m-1)}_{b,i} = x^{(m)}_{b,i+7} \oplus x^{(m)}_{c,i} \tag{5}$$

$$x'^{(m-1)}_{c,i} = x^{(m)}_{c,i} \oplus x^{(m)}_{d,i} \oplus \Theta_i(x'^{(m-1)}_c, x^{(m)}_d) \tag{6}$$

$$x'^{(m-1)}_{d,i} = x^{(m)}_{a,i} \oplus x^{(m)}_{d,i+8} \tag{7}$$

$$x'^{(m-1)}_{a,i} = x^{(m)}_{a,i} \oplus x^{(m)}_{b,i+7} \oplus x^{(m)}_{c,i} \oplus \Theta_i(x'^{(m-1)}_a, x'^{(m-1)}_b) \tag{8}$$

$$x^{(m-1)}_{b,i} = \mathcal{L}^{(m)}_{b,i} \oplus \Theta_i(x'^{(m-1)}_c, x^{(m)}_d) \tag{9}$$

$$x^{(m-1)}_{c,i} = \mathcal{L}^{(m)}_{c,i} \oplus \Theta_i(x'^{(m-1)}_c, x^{(m)}_d) \oplus \Theta_i(x^{(m-1)}_c, x'^{(m-1)}_d) \tag{10}$$

$$x^{(m-1)}_{d,i} = \mathcal{L}^{(m)}_{d,i} \oplus \Theta_i(x'^{(m-1)}_a, x'^{(m-1)}_b) \tag{11}$$

$$\begin{aligned}
x^{(m-1)}_{a,i} = {} & \mathcal{L}^{(m)}_{a,i} \oplus \Theta_i(x'^{(m-1)}_a, x'^{(m-1)}_b) \oplus \\
& \Theta_i(x'^{(m-1)}_c, x^{(m)}_d) \oplus \Theta_i(x^{(m-1)}_a, x^{(m-1)}_b)
\end{aligned} \tag{12}$$

where

$$\mathcal{L}^{(m)}_{a,i} = x^{(m)}_{a,i} \oplus x^{(m)}_{b,i+7} \oplus x^{(m)}_{b,i+19} \oplus x^{(m)}_{c,i+12} \oplus x^{(m)}_{d,i} \tag{13}$$

$$\mathcal{L}^{(m)}_{b,i} = x^{(m)}_{b,i+19} \oplus x^{(m)}_{c,i} \oplus x^{(m)}_{c,i+12} \oplus x^{(m)}_{d,i} \tag{14}$$

$$\mathcal{L}^{(m)}_{c,i} = x^{(m)}_{a,i} \oplus x^{(m)}_{c,i} \oplus x^{(m)}_{d,i} \oplus x^{(m)}_{d,i+8} \tag{15}$$

$$\mathcal{L}^{(m)}_{d,i} = x^{(m)}_{a,i} \oplus x^{(m)}_{a,i+16} \oplus x^{(m)}_{b,i+7} \oplus x^{(m)}_{c,i} \oplus x^{(m)}_{d,i+24} \tag{16}$$

Lemma 1. *It holds that* $x^{(m-1)}_{l,0} = \mathcal{L}^{(m)}_{l,0}$, *for* $l \in \{a, b, c, d\}$.

Proof. This result follows directly from Eqs. (9)–(12) by using the fact that $\Theta_0(x, y) = 0$. □

From this equations, we can derive the following lemma:

Lemma 2. *(Lemma 3 of [9]) Let*

$$\begin{aligned}
\Delta A^{(m)} &= \Delta x_{\alpha,0}^{(m)} \oplus \Delta x_{\beta,7}^{(m)} \oplus \Delta x_{\beta,19}^{(m)} \oplus \Delta x_{\gamma,12}^{(m)} \oplus \Delta x_{\delta,0}^{(m)} \\
\Delta B^{(m)} &= \Delta x_{\beta,19}^{(m)} \oplus \Delta x_{\gamma,0}^{(m)} \oplus \Delta x_{\gamma,12}^{(m)} \oplus \Delta x_{\delta,0}^{(m)} \\
\Delta C^{(m)} &= \Delta x_{\delta,0}^{(m)} \oplus \Delta x_{\gamma,0}^{(m)} \oplus \Delta x_{\delta,8}^{(m)} \oplus \Delta x_{\alpha,0}^{(m)} \\
\Delta D^{(m)} &= \Delta x_{\delta,24}^{(m)} \oplus \Delta x_{\alpha,16}^{(m)} \oplus \Delta x_{\alpha,0}^{(m)} \oplus \Delta x_{\gamma,0}^{(m)} \oplus \Delta x_{\beta,7}^{(m)}
\end{aligned}$$

After m rounds of ChaCha, the following holds:

$$\left|\varepsilon_{(A(m))}\right| = \left|\varepsilon_{\left(x_{\alpha,0}^{(m-1)}\right)}\right|, \left|\varepsilon_{\left(B^{(m)}\right)}\right| = \left|\varepsilon_{\left(x_{\beta,0}^{(m-1)}\right)}\right|$$

$$\left|\varepsilon_{\left(C^{(m)}\right)}\right| = \left|\varepsilon_{\left(x_{\gamma,0}^{(m-1)}\right)}\right|, \left|\varepsilon_{\left(D^{(m)}\right)}\right| = \left|\varepsilon_{\left(x_{\delta,0}^{(m-1)}\right)}\right|$$

The tuples $(\alpha, \beta, \gamma, \delta)$ vary depending on whether m is odd or even.

- *Case I. m is odd:*

$$(\alpha, \beta, \gamma, \delta) \in \{(0,4,8,12), (1,5,9,13), (2,6,10,14), (3,7,11,15)\}.$$

- *Case II. m is even:*

$$(\alpha, \beta, \gamma, \delta) \in \{(0,5,10,15), (1,6,11,12), (2,7,8,13), (3,4,9,14)\}.$$

Proof. See [9]. □

Lemma 3. *(Lemma 9 of [9]) For one active input bit in round $m-1$ and multiple active output bits in round m, the following holds for $i > 0$.*

$$\begin{aligned}
x_{b,i}^{(m-1)} &= \mathcal{L}_{b,i}^{(m)} \oplus x_{d,i-1}^{(m)}, && \text{w.p. } \tfrac{1}{2}\left(1+\tfrac{1}{2}\right) \\
x_{a,i}^{(m-1)} &= \mathcal{L}_{a,i}^{(m)} \oplus x_{b,i+18}^{(m)} \oplus x_{c,i+11}^{(m)} \oplus x_{d,i-2}^{(m)} \oplus x_{d,i+6}^{(m)}, && \text{w.p. } \tfrac{1}{2}\left(1+\tfrac{1}{2^4}\right) \\
x_{c,i}^{(m-1)} &= \mathcal{L}_{c,i}^{(m)} \oplus x_{a,i-1}^{(m)} \oplus x_{d,i+7}^{(m)} \oplus x_{d,i-1}^{(m)}, && \text{w.p. } \tfrac{1}{2}\left(1+\tfrac{1}{2^2}\right) \\
x_{d,i}^{(m-1)} &= \mathcal{L}_{d,i}^{(m)} \oplus x_{c,i-1}^{(m)} \oplus x_{b,i+6}^{(m)}, && \text{w.p. } \tfrac{1}{2}\left(1+\tfrac{1}{2}\right)
\end{aligned}$$

Proof. See [9]. □

Finally, using Lemma 2 and Lemma 3, it is possible to find linear approximations for two rounds of ChaCha.

Lemma 4. *(Lemma 10 of [9]) The following holds with probability $\frac{1}{2}\left(1+\frac{1}{2}\right)$*

$$\begin{aligned}
x_{11,0}^{(3)} = {} & x_0^{(5)}[0,8,16,24] \oplus x_{1,0}^{(5)} \oplus x_{3,0}^{(5)} \oplus x_{4,7}^{(5)} \oplus x_4^{(5)}[14,15] \oplus x_5^{(5)}[7,19] \oplus \\
& x_8^{(5)}[0,7,8] \oplus x_{9,12}^{(5)} \oplus x_{11,0}^{(5)} \oplus x_{12}^{(5)}[0,24] \oplus x_{13,0}^{(5)} \oplus x_{15}^{(5)}[0,8].
\end{aligned}$$

Proof. See [9]. □

Recently, Coutinho and Souza [10] found linear approximations with fewer terms using the same techniques.

Lemma 5. *(Lemma 5 of [10]) When m is odd, each of the following also holds with probability $\frac{1}{2}(1+\frac{1}{2})$*

$$\begin{aligned}
x_{0,0}^{(m-2)} \oplus x_{5,0}^{(m-2)} &= x_{0,0}^{(m)} \oplus x_{2,0}^{(m)} \oplus x_{4,7}^{(m)} \oplus x_{4,19}^{(m)} \oplus x_{5,26}^{(m)} \oplus x_{8,12}^{(m)} \oplus x_{9,7}^{(m)} \oplus \\
&\quad x_{9,19}^{(m)} \oplus x_{10,0}^{(m)} \oplus x_{12,0}^{(m)} \oplus x_{13,6}^{(m)} \oplus x_{13,7}^{(m)} \oplus x_{14,0}^{(m)} \oplus x_{14,8}^{(m)} \\
x_{1,0}^{(m-2)} \oplus x_{6,0}^{(m-2)} &= x_{1,0}^{(m)} \oplus x_{3,0}^{(m)} \oplus x_{5,7}^{(m)} \oplus x_{5,19}^{(m)} \oplus x_{6,26}^{(m)} \oplus x_{9,12}^{(m)} \oplus x_{10,7}^{(m)} \oplus \\
&\quad x_{10,19}^{(m)} \oplus x_{11,0}^{(m)} \oplus x_{13,0}^{(m)} \oplus x_{14,6}^{(m)} \oplus x_{14,7}^{(m)} \oplus x_{15,0}^{(m)} \oplus x_{15,8}^{(m)} \\
x_{2,0}^{(m-2)} \oplus x_{7,0}^{(m-2)} &= x_{0,0}^{(m)} \oplus x_{2,0}^{(m)} \oplus x_{6,7}^{(m)} \oplus x_{6,19}^{(m)} \oplus x_{7,26}^{(m)} \oplus x_{8,0}^{(m)} \oplus x_{10,12}^{(m)} \oplus \\
&\quad x_{11,7}^{(m)} \oplus x_{11,19}^{(m)} \oplus x_{12,0}^{(m)} \oplus x_{12,8}^{(m)} \oplus x_{14,0}^{(m)} \oplus x_{15,6}^{(m)} \oplus x_{15,7}^{(m)} \\
x_{3,0}^{(m-2)} \oplus x_{4,0}^{(m-2)} &= x_{1,0}^{(m)} \oplus x_{3,0}^{(m)} \oplus x_{4,26}^{(m)} \oplus x_{7,7}^{(m)} \oplus x_{7,19}^{(m)} \oplus x_{8,7}^{(m)} \oplus x_{8,19}^{(m)} \oplus \\
&\quad x_{9,0}^{(m)} \oplus x_{11,12}^{(m)} \oplus x_{12,6}^{(m)} \oplus x_{12,7}^{(m)} \oplus x_{13,0}^{(m)} \oplus x_{13,8}^{(m)} \oplus x_{15,0}^{(m)}
\end{aligned}$$

Proof. See [10]. □

In [9], the authors showed that using as $\mathcal{ID}$ a single bit at $x_{13,13}^{(0)}$ and $\mathcal{OD}$ at $x_{11,0}^{(3)}$, it is possible to obtain $\varepsilon_d = -0.0272 \approx -\frac{1}{2^{5.2}}$, experimentally. And from Lemma 2 it is possible to extend to a 4-round differential-linear correlation with $\varepsilon_L = 1$ when the $\mathcal{OD}$ is $x_{1,0}^{(4)} \oplus x_{11,0}^{(4)} \oplus x_{12,8}^{(4)} \oplus x_{12,0}^{(4)}$. Further, it is possible to extend to a 5-round differential-linear correlation using the last equation from Lemma 4 with probability $\frac{1}{2}\left(1+\frac{1}{2}\right)$. This gives a total differential-linear 5th round correlation of $\varepsilon_d \cdot \varepsilon_L^2 \approx -0.0068 = -\frac{1}{2^{7.2}}$. This leads to a 5 round distinguisher with complexity approximately 2^{16}.

Extending the linear approximation for 3 rounds comes at a cost. As discussed prior to the above lemma, for ChaCha, setting $i = 0$ in Lemma 2 allows linear approximation of probability 1 for LSB variables. The cost is thus determined by the non LSB variables. A simple count of the non LSB variables in the form (Variable Type, # non LSB occurrence) gives $(x_a, 3), (x_b, 5), (x_c, 3)$, and $(x_d, 2)$. Now, using the probabilities of Lemma 3 and Lemma 4, the linear correlation is $\varepsilon_L = 1/2^{1+3\cdot4+5\cdot1+3\cdot2+2\cdot1} = 2^{-26}$. This leads to a 6 round correlation of $\varepsilon_L^2 \varepsilon_d \approx \frac{1}{2^{57.2}}$. The distinguisher for this correlation has a complexity of 2^{116}.

In [10], the authors used Lemma 5 to derive a distinguisher for 6 rounds. To do that, they found a differential with correlation $\varepsilon_d = 0.00048$ for $(a, b) = (3, 4)$ when the input difference is given by $\Delta x_{14,6}^{(0)} = 1$, and 0 for all remaining bits. Therefore, expanding for 6 rounds from Lemma 5 with weights $4, 1, 2, 1$ for x_a, x_b, x_c and x_d, respectively, they got $\varepsilon_L = 1/2^{1+0\cdot4+3\cdot1+3\cdot2+3\cdot1} = 2^{-13}$. Then we have $\varepsilon_d \varepsilon_L^2 \approx 2^{-37.02}$, which leads to an attack against 6 rounds of ChaCha with complexity 2^{75}. This is the currently best known 6 round attack on ChaCha.

3 Improved Linear Approximations for ARX-Based Ciphers

The challenge of finding good linear approximations in ARX-based designs comes from the addition operation which is responsible for the non-linearity of the design. In 2003, Wallén [30] published a very important paper where a mathematical framework for finding linear approximations of addition modulo 2^n was developed. Since then, several authors used these technique to find linear approximations in ARX-based designs [9].

Therefore, as before, let $\Theta(x, y) = x \oplus y \oplus (x + y)$ be the carry function of the sum $x + y$. Define $\Theta_i(x, y)$ as the i-th bit of $\Theta(x, y)$. By definition, we have $\Theta_0(x, y) = 0$. Using Theorem 3 of [30], we can generate all possible linear approximations with a given correlation. In particular, we will use the following linear approximations:

$$\Pr(\Theta_i(x, y) = y_{i-1}) = \frac{1}{2}\left(1 + \frac{1}{2}\right), i > 0. \tag{17}$$

$$\Pr(\Theta_i(x, y) \oplus \Theta_{i-1}(x, y) = 0) = \frac{1}{2}\left(1 + \frac{1}{2}\right), i > 0. \tag{18}$$

In previous works of cryptanalysis of ARX ciphers, authors concentrated in finding approximations for particular bits in one round and then repeating the same equations to expand the linear approximation to further rounds (see [9] and [10] for some examples). However, by combining Eqs. 17 and 18 when attacking ARX ciphers we can create a strategy to improve linear approximations when considering more rounds. The main idea is that when using Eq. 17 in one round we will create consecutive terms that can be expanded together using Eq. 18.

For example, consider the sum $z = x + y$. If we want a linear approximation for the bit z_7, we can use Eq. 17 to obtain $z_7 = x_7 \oplus y_7 \oplus \Theta_7(x, y) = x_7 \oplus y_7 \oplus y_6$ with probability 0.75. Since the XOR operation will not change the indexes and the rotation will probably keep y_6 and y_7 adjacent, we can use Eq. 18 in the subsequent round to cancel out the non-linear terms rather than expanding them, leading to a linear equation with higher correlation and fewer terms to be expanded further. Next, we will use this technique to find new linear approximations for ChaCha.

3.1 Linear Approximations for the Quarter Round Function

The first step is to find linear approximations for the QRF of ChaCha. Of course, we already know some of them from previous works (Sect. 2.3). However, here we will consider adjacent bits and several other combinations that cancel out non-linear terms or use Eq. (18). At first glance, these results may seem innocuous, but latter they will prove themselves useful when deriving linear approximations for multiple rounds of ChaCha.

We start with a better linear approximation for $x_{a,i}^{(m-1)}$.

Lemma 6. *The following holds for $i > 0$*

$$x_{a,i}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus x_{b,i+6}^{(m)} \oplus x_{b,i+18}^{(m)} \oplus x_{c,i+11}^{(m)} \oplus x_{d,i-1}^{(m)}, \text{ w.p. } \tfrac{1}{2}\left(1 + \tfrac{1}{2^3}\right).$$

Proof. From Eq. (12) we have

$$x_{a,i}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus \Theta_i(x_a'^{(m-1)}, x_b'^{(m-1)}) \oplus \Theta_i(x_c'^{(m-1)}, x_d^{(m)}) \oplus \Theta_i(x_a^{(m-1)}, x_b^{(m-1)}).$$

Using Eq. (17) and the Piling-up Lemma we can write

$$x_{a,i}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus x_{b,i-1}'^{(m-1)} \oplus \Theta_i(x_c'^{(m-1)}, x_d^{(m)}) \oplus x_{b,i-1}^{(m-1)},$$

with probability $\frac{1}{2}\left(1 + \frac{1}{2^2}\right)$. Using Eq. (9) we get

$$x_{a,i}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus x_{b,i-1}'^{(m-1)} \oplus \Theta_i(x_c'^{(m-1)}, x_d^{(m)}) \oplus \mathcal{L}_{b,i-1}^{(m)} \oplus \Theta_{i-1}(x_c'^{(m-1)}, x_d^{(m)}).$$

Using the approximation of Eq. (18) and the Piling-up Lemma we can write

$$x_{a,i}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus x_{b,i-1}'^{(m-1)} \oplus \mathcal{L}_{b,i-1}^{(m)},$$

with probability $\frac{1}{2}\left(1 + \frac{1}{2^3}\right)$. Finally, using Eqs. (5) and (14) we get

$$x_{a,i}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus x_{b,i+6}^{(m)} \oplus x_{b,i+18}^{(m)} \oplus x_{c,i+11}^{(m)} \oplus x_{d,i-1}^{(m)},$$

which completes the proof. □

Lemma 7. *For two active input bits in round $m-1$ and multiple active output bits in round m, the following holds for $i > 0$*

$$x_{\lambda,i}^{(m-1)} \oplus x_{\lambda,i-1}^{(m-1)} = \mathcal{L}_{\lambda,i}^{(m)} \oplus \mathcal{L}_{\lambda,i-1}^{(m)}, \text{ w.p. } \frac{1}{2}\left(1 + \frac{1}{2^\sigma}\right),$$

where $(\lambda, \sigma) \in \{(a,3),(b,1),(c,2),(d,1)\}$.

Proof. This proof follows directly from Eqs. (9)–(12) using the approximation of Eq. (18) and the Piling-up Lemma. □

Lemma 8. *Suppose that $(\lambda, \sigma) \in \{(i, i-2),(i-1,i-1)\}$, $i > 1$. Then for three active input bits in round $m-1$ and multiple active output bits in round m, the following holds*

$$x_{b,\lambda}^{(m-1)} \oplus x_{c,i}^{(m-1)} \oplus x_{c,i-1}^{(m-1)} = \mathcal{L}_{b,i-1}^{(m)} \oplus \mathcal{L}_{c,i}^{(m)} \oplus \mathcal{L}_{c,i-1}^{(m)} \oplus x_{d,\sigma}^{(m)}, \text{ w.p. } \frac{1}{2}\left(1 + \frac{1}{2^2}\right).$$

Proof. Using Eq. (9) and Eq. (10) we get

$$\begin{aligned} x_{b,\lambda}^{(m-1)} \oplus x_{c,i}^{(m-1)} \oplus x_{c,i-1}^{(m-1)} = {} & \mathcal{L}_{b,\lambda}^{(m)} \oplus \mathcal{L}_{c,i}^{(m)} \oplus \mathcal{L}_{c,i-1}^{(m)} \oplus \Theta_\lambda(x_c'^{(m-1)}, x_d^{(m)}) \oplus \\ & \Theta_i(x_c'^{(m-1)}, x_d^{(m)}) \oplus \Theta_i(x_c^{(m-1)}, x_d'^{(m-1)}) \oplus \\ & \Theta_{i-1}(x_c'^{(m-1)}, x_d^{(m)}) \oplus \Theta_{i-1}(x_c^{(m-1)}, x_d'^{(m-1)}). \end{aligned}$$

Canceling out common factors and using the approximation of Eq. (18) we can write

$$x_{b,\lambda}^{(m-1)} \oplus x_{c,i}^{(m-1)} \oplus x_{c,i-1}^{(m-1)} = \mathcal{L}_{b,i}^{(m)} \oplus \mathcal{L}_{c,i}^{(m)} \oplus \mathcal{L}_{c,i-1}^{(m)} \oplus \Theta_{\sigma+1}(x_c'^{(m-1)}, x_d^{(m)}).$$

with probability $\frac{1}{2}\left(1+\frac{1}{2}\right)$. Using Eq. (17) we get

$$x_{b,\lambda}^{(m-1)} \oplus x_{c,i}^{(m-1)} \oplus x_{c,i-1}^{(m-1)} = \mathcal{L}_{b,i}^{(m)} \oplus \mathcal{L}_{c,i}^{(m)} \oplus \mathcal{L}_{c,i-1}^{(m)} \oplus x_{d,\sigma}^{(m)},$$

with probability $\frac{1}{2}\left(1+\frac{1}{2^2}\right)$. □

Lemma 9. *For multiple active input bits in round $m-1$ and multiple active output bits in round m, the following linear approximations hold for ChaCha with probability $\frac{1}{2}\left(1+\frac{1}{2^k}\right)$:*

$$x_{b,i}^{(m-1)} \oplus x_{c,i}^{(m-1)} = \mathcal{L}_{b,i}^{(m)} \oplus \mathcal{L}_{c,i}^{(m)} \oplus x_{a,i-1}^{(m)} \oplus x_{d,i+7}^{(m)} \qquad k=1, i>0 \qquad (19)$$

$$x_{a,i}^{(m-1)} \oplus x_{b,i}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{b,i-1}^{(m)} \oplus \mathcal{L}_{b,i}^{(m)} \oplus x_{b,i+6}^{(m)} \oplus x_{c,i-1}^{(m)} \oplus x_{d,i-2}^{(m)} \qquad k=3, i>1 \qquad (20)$$

$$x_{a,1}^{(m-1)} \oplus x_{b,1}^{(m-1)} = \mathcal{L}_{a,1}^{(m)} \oplus \mathcal{L}_{b,0}^{(m)} \oplus \mathcal{L}_{b,1}^{(m)} \oplus x_{b,7}^{(m)} \oplus x_{c,0}^{(m)} \qquad k=2 \qquad (21)$$

$$x_{a,i}^{(m-1)} \oplus x_{c,i}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{b,i-1}^{(m)} \oplus \mathcal{L}_{c,i}^{(m)} \oplus x_{a,i-1}^{(m)} \oplus x_{b,i+6}^{(m)} \oplus x_{c,i-1}^{(m)} \oplus x_{d,i-2}^{(m)} \oplus x_{d,i+7}^{(m)} \qquad k=4, i>1 \qquad (22)$$

$$x_{a,1}^{(m-1)} \oplus x_{c,1}^{(m-1)} = \mathcal{L}_{a,1}^{(m)} \oplus \mathcal{L}_{b,0}^{(m)} \oplus \mathcal{L}_{c,1}^{(m)} \oplus x_{a,0}^{(m)} \oplus x_{b,7}^{(m)} \oplus x_{c,0}^{(m)} \oplus x_{d,8}^{(m)} \qquad k=3 \qquad (23)$$

$$x_{a,i}^{(m-1)} \oplus x_{d,i}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{d,i}^{(m)} \oplus \mathcal{L}_{b,i-1}^{(m)} \qquad k=2, i>1 \qquad (24)$$

$$x_{a,i-1}^{(m-1)} \oplus x_{a,i}^{(m-1)} \oplus x_{c,i}^{(m-1)} = \mathcal{L}_{a,i-1}^{(m)} \oplus \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{c,i}^{(m)} \oplus x_{d,i-2}^{(m)} \oplus x_{a,i-1}^{(m)} \oplus x_{d,i+7}^{(m)} \qquad k=4, i>1 \qquad (25)$$

$$x_{a,i}^{(m-1)} \oplus x_{a,i-1}^{(m-1)} \oplus x_{b,i}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{a,i-1}^{(m)} \oplus \mathcal{L}_{b,i}^{(m)} \oplus x_{d,i-2}^{(m)}, \qquad k=3, i>1 \qquad (26)$$

$$x_{b,i-1}^{(m-1)} \oplus x_{a,i}^{(m-1)} \oplus x_{d,i}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{d,i}^{(m)} \oplus x_{d,i-1}^{(m)}, \qquad k=2, i>1 \qquad (27)$$

$$x_{b,i-1}^{(m-1)} \oplus x_{b,i}^{(m-1)} \oplus x_{c,i-1}^{(m-1)} \oplus x_{c,i}^{(m-1)} = \mathcal{L}_{b,i-1}^{(m)} \oplus \mathcal{L}_{b,i}^{(m)} \oplus \mathcal{L}_{c,i-1}^{(m)} \oplus \mathcal{L}_{c,i}^{(m)}, \qquad k=1, i>1 \qquad (28)$$

$$x_{a,i}^{(m-1)} \oplus x_{a,i-1}^{(m-1)} \oplus x_{b,i}^{(m-1)} \oplus x_{c,i-1}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{a,i-1}^{(m)} \oplus \mathcal{L}_{b,i}^{(m)} \oplus \mathcal{L}_{c,i-1}^{(m)} \oplus x_{a,i-2}^{(m)} \oplus x_{d,i+6}^{(m)}, \qquad k=3, i>1 \qquad (29)$$

$$x_{a,i}^{(m-1)} \oplus x_{a,i-1}^{(m-1)} \oplus x_{c,i-1}^{(m-1)} \oplus x_{d,i}^{(m-1)} \oplus x_{d,i-1}^{(m-1)} = \mathcal{L}_{a,i-1}^{(m)} \oplus \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{c,i-1}^{(m)} \oplus \mathcal{L}_{d,i-1}^{(m)} \oplus \mathcal{L}_{d,i}^{(m)} \oplus x_{d,i-1}^{(m)} \oplus x_{a,i-2}^{(m)} \oplus x_{d,i+6}^{(m)}, \qquad k=3, i>2 \qquad (30)$$

Proof. The proof for each equation follows the same basic steps: (1) cancel common factors; (2) cancel adjacent non-linear terms using Eq. (18), updating the probability using the Piling-Up Lemma; (3) substitute the remaining non-linear

terms using Eq. (17), updating the probability using the Piling-Up Lemma. For completeness, we list all proofs in Appendix A. □

3.2 Linear Approximations for Multiple Rounds of ChaCha

In this section, we use the proposed technique to construct several new linear approximations for the stream cipher ChaCha which will prove useful to construct better distinguishers. We developed a program (available in https://github.com/MurCoutinho/cryptanalysisChaCha.git) that makes the process of finding linear approximations partly automatic. Our program is capable of expanding the equations and, after statistically verifying the correlation, it outputs the resulting linear approximation in LaTeXcode.

We start using the result of Coutinho and Souza [10]. We will only consider the equation for $x_{3,0}^{(3)} \oplus x_{4,0}^{(3)}$ of Lemma 5 but the same reasoning could be applied to any other equation in that lemma. Then, we have

$$\begin{aligned} x_{3,0}^{(3)} \oplus x_{4,0}^{(3)} = x_{1,0}^{(5)} \oplus x_{3,0}^{(5)} \oplus x_{4,26}^{(5)} \oplus x_{7,7}^{(5)} \oplus x_{7,19}^{(5)} \oplus x_{8,7}^{(5)} \oplus x_{8,19}^{(5)} \oplus \\ x_{9,0}^{(5)} \oplus x_{11,12}^{(5)} \oplus x_{12,6}^{(5)} \oplus x_{12,7}^{(5)} \oplus x_{13,0}^{(5)} \oplus x_{13,8}^{(5)} \oplus x_{15,0}^{(5)} \end{aligned} \tag{31}$$

with probability $\frac{1}{2}\left(1 + \frac{1}{2}\right)$.

As presented in Sect. 2.3, to expand the equation to the 6-th round, we could use only Lemma 3 as proposed in [9]. In this case, we have weights $4, 1, 2, 1$ for x_a, x_b, x_c and x_d, respectively, and a count of $(x_a, 0)$, $(x_b, 3)$, $(x_c, 3)$ e $(x_d, 3)$. Thus, the linear correlation is $\varepsilon_L = 1/2^{1+0\cdot4+3\cdot1+3\cdot2+3\cdot1} = 2^{-13}$. However, we can do better with the new technique proposed in Sect. 3. This will lead us to the following lemma

Lemma 10. *The following linear approximation holds with probability* $\frac{1}{2}\left(1 + \frac{1}{2^8}\right)$

$$\begin{aligned} x_{3,0}^{(3)} \oplus x_{4,0}^{(3)} = x_0^{(6)}[0,16] \oplus x_1^{(6)}[0,6,7,11,12,22,23] \oplus x_2^{(6)}[0,6,7,8,16,18, \\ 19,24] \oplus x_4^{(6)}[7,13,19] \oplus x_5^{(6)}[7] \oplus x_6^{(6)}[7,13,14,19] \oplus \\ x_7^{(6)}[6,7,14,15,26] \oplus x_8^{(6)}[0,7,8,19,31] \oplus x_9^{(6)}[0,6,12,26] \oplus \\ x_{10}^{(6)}[0] \oplus x_{11}^{(6)}[6,7] \oplus x_{12}^{(6)}[0,11,12,19,20,30,31] \oplus \\ x_{13}^{(6)}[0,14,15,24,26,27] \oplus x_{14}^{(6)}[8,25,26] \oplus x_{15}^{(6)}[24]. \end{aligned}$$

Proof. First, from Eq. (31) we can use Lemma 1 to replace $x_{1,0}^{(5)}$, $x_{3,0}^{(5)}$, $x_{9,0}^{(5)}$, $x_{13,0}^{(5)}$, $x_{15,0}^{(5)}$ by $\mathcal{L}_{1,0}^{(6)}$, $\mathcal{L}_{3,0}^{(6)}$, $\mathcal{L}_{9,0}^{(6)}$, $\mathcal{L}_{13,0}^{(6)}$, $\mathcal{L}_{15,0}^{(6)}$ with probability 1. Next, note that, since we are transitioning from round 5 to 6, we have

$$(a, b, c, d) \in \{(0, 5, 10, 15), (1, 6, 11, 12), (2, 7, 8, 13), (3, 4, 9, 14)\}.$$

We have already considered the case $(a, b, c, d) = (0, 5, 10, 15)$. Then we still have 3 cases left to consider.

– **Case 1:** When $(a, b, c, d) = (1, 6, 11, 12)$, we have the factors $x^{(5)}_{11,12}$, $x^{(5)}_{12,6}$, $x^{(5)}_{12,7}$. Then we can use Lemma 3 and Lemma 7 in order to get

$$\Pr\left(x^{(5)}_{11,12} = \mathcal{L}^{(6)}_{11,12} \oplus x^{(6)}_{1,11} \oplus x^{(6)}_{12,19} \oplus x^{(6)}_{12,11}\right) = \frac{1}{2}\left(1 + \frac{1}{2^2}\right)$$

and

$$\Pr\left(x^{(5)}_{12,7} \oplus x^{(5)}_{12,6} = \mathcal{L}^{(6)}_{12,7} \oplus \mathcal{L}^{(6)}_{12,6}\right) = \frac{1}{2}\left(1 + \frac{1}{2}\right).$$

– **Case 2:** If $(a, b, c, d) = (2, 7, 8, 13)$, we have the factors $x^{(5)}_{7,7}$, $x^{(5)}_{7,19}$, $x^{(5)}_{8,7}$, $x^{(5)}_{8,19}$, $x^{(5)}_{13,8}$ and we can use Lemma 3 and Eq. (19) of Lemma 9 to get

$$\Pr\left(x^{(5)}_{13,8} = \mathcal{L}^{(6)}_{13,8} \oplus x^{(6)}_{8,7} \oplus x^{(6)}_{7,i+6}\right) = \frac{1}{2}\left(1 + \frac{1}{2}\right),$$

$$\Pr\left(x^{(5)}_{7,7} \oplus x^{(5)}_{8,7} = \mathcal{L}^{(6)}_{7,7} \oplus \mathcal{L}^{(6)}_{8,7} \oplus x^{(6)}_{2,6} \oplus x^{(6)}_{13,14}\right) = \frac{1}{2}\left(1 + \frac{1}{2}\right),$$

$$\Pr\left(x^{(5)}_{7,19} \oplus x^{(5)}_{8,19} = \mathcal{L}^{(6)}_{7,19} \oplus \mathcal{L}^{(6)}_{8,19} \oplus x^{(6)}_{2,18} \oplus x^{(6)}_{13,26}\right) = \frac{1}{2}\left(1 + \frac{1}{2}\right).$$

– **Case 3:** When considering $(a, b, c, d) = (3, 4, 9, 14)$, we have $x^{(5)}_{4,26}$ and we can use Lemma 3 to obtain

$$\Pr\left(x^{(5)}_{4,26} = \mathcal{L}^{(6)}_{4,26} \oplus x^{(6)}_{14,26}\right) = \frac{1}{2}\left(1 + \frac{1}{2}\right).$$

By the Piling-up Lemma, we have that all these changes result in a probability of $\frac{1}{2}\left(1 + \frac{1}{2^8}\right)$. Expanding the linear terms using Eqs. (13)–(16) and canceling out common factors completes the proof. □

Computational Result 1 *The linear approximation of Lemma 10 holds computationally with* $\varepsilon_{L_0} = 0.006942 \approx 2^{-7.17}$. *This correlation was verified using* 2^{38} *random samples.*

In [9], the authors remarked that an expansion of this method to 7 rounds would be unlikely to be useful. Indeed, if we only apply Lemma 3 (which are the linear approximations proposed by Choudhuri and Maitra) we would have $(x_a, 14)$, $(x_b, 13)$, $(x_c, 9)$, $(x_d, 15)$. Therefore, the aggregated correlation would be $\varepsilon_L = 1/2^{7+14\cdot4+13\cdot1+9\cdot2+15\cdot1} = 2^{-109}$. Thus, using this linear expansion in a differential-linear attack would lead to a distinguisher with complexity no less then 2^{436}. However, using our new linear approximations we can get a much better result.

Lemma 11. *The following linear approximation holds with probability* $\frac{1}{2}\left(1+\frac{1}{2^{55}}\right)$

$$\begin{aligned} x_{3,0}^{(3)} \oplus x_{4,0}^{(3)} = {} & x_0^{(7)}[0,3,4,7,8,11,12,14,15,18,20,27,28] \oplus x_1^{(7)}[0,5,7,8,10, \\ & 11,14,15,16,22,23,24,25,27,30,31] \oplus x_2^{(7)}[6,7,9,10,16,18,19, \\ & 25,26] \oplus x_3^{(7)}[6,7,8,24] \oplus x_4^{(7)}[0,2,3,5,18,22,23,27] \oplus x_5^{(7)}[1,2, \\ & 9,10,13,14,18,21,22,25,29,30] \oplus x_6^{(7)}[2,3,5,7,10,11,13,14,19, \\ & 22,23,27,30,31] \oplus x_7^{(7)}[1,2,13,25,26,30,31] \oplus x_8^{(7)}[8,11,13,20, \\ & 25,27,28,30,31] \oplus x_9^{(7)}[2,3,6,7,14,15,18,27] \oplus x_{10}^{(7)}[0,3,4,8,12, \\ & 13,14,18,20,27,28,30] \oplus x_{11}^{(7)}[6,14,15,18,19,23,24,27] \oplus \\ & x_{12}^{(7)}[3,4,6,11,13,22,23,24,26,27,30,31] \oplus x_{13}^{(7)}[1,2,6,7,8,10, \\ & 11,13,14,16,18,20,22,23,24,25,26] \oplus x_{14}^{(7)}[0,6,13,14,15,16, \\ & 23,24] \oplus x_{15}^{(7)}[16,25,26]. \end{aligned}$$

Proof. If we start from Lemma 10 then we want to expand the equation one more round. To do so, first note that since we are transitioning from round 6 to 7, we have $(a,b,c,d) \in \{(0,4,8,12),(1,5,9,13),(2,6,10,14),(3,7,11,15)\}$. Therefore, we can divide the factors of the equation in 4 distinct groups:

- Group I - $x_0^{(6)}[0,16], x_4^{(6)}[7,13,19], x_8^{(6)}[0,7,8,19,31]$, $x_{12}^{(6)}[0,11,12,19,20,30,31]$.
- Group II - $x_1^{(6)}[0,6,7,11,12,22,23]$, $x_5^{(6)}[7]$, $x_9^{(6)}[0,6,12,26]$, $x_{13}^{(6)}[0,14,15,24,26,27]$.
- Group III - $x_2^{(6)}[0,6,7,8,16,18,19,24], x_6^{(6)}[7,13,14,19], x_{10}^{(6)}[0], x_{14}^{(6)}[8,25,26]$.
- Group IV - $x_7^{(6)}[6,7,14,15,26]$, $x_{11}^{(6)}[6,7], x_{15}^{(6)}[24]$.

The procedure to expand and compute the correlation is similar to that in the proof of Lemma 10. To simplify the notation we will compute the probability given by the Piling-up Lemma by summing values k where the probability of a particular linear equation will be given by $\frac{1}{2}\left(1+\frac{1}{2^k}\right)$.

In Group I, the factors $x_{0,0}^{(6)}, x_{8,0}^{(6)}, x_{12,0}^{(6)}$ can be expanded using Lemma 1 with probability 1. Next, we can combine the following factors: $x_{4,7}^{(6)}, x_{8,7}^{(6)}, x_{8,8}^{(6)}$ using Lemma 8 ($k=2$); $x_{4,19}^{(6)}, x_{8,19}^{(6)}$ using Eq. (19) of Lemma 9 ($k=1$); $x_{12,11}^{(6)}, x_{12,12}^{(6)}$ using Lemma 7 with ($k=1$); $x_{12,19}^{(6)}, x_{12,20}^{(6)}$ using Lemma 7 with ($k=1$); $x_{12,30}^{(6)}, x_{12,31}^{(6)}$ using Lemma 7 with ($k=1$). Finally, it remains some single terms to be expanded: $x_{0,16}^{(6)}$ using Lemma 6 ($k=3$); $x_{4,13}^{(6)}$ using Lemma 3 ($k=1$); $x_{8,31}^{(6)}$ using Lemma 3 ($k=2$). By the Piling-up Lemma, we can combine these linear relations to obtain

$$\begin{aligned} & x_0^{(6)}[0,16] \oplus x_4^{(6)}[7,13,19] \oplus x_8^{(6)}[0,7,8,19,31] \oplus x_{12}^{(6)}[0,11,12,19,20,30,31] = \\ & x_0^{(7)}[0,3,4,7,8,11,12,14,15,18,20,27,28] \oplus x_4^{(7)}[0,2,3,5,18,22,23,27] \oplus \\ & x_8^{(7)}[8,11,13,20,25,27,28,30,31] \oplus x_{12}^{(7)}[3,4,6,11,13,22,23,24,26,27,30,31] \end{aligned} \tag{32}$$

with probability $\frac{1}{2}\left(1+\frac{1}{2^{12}}\right)$.

In Group II, the factors $x^{(6)}_{1,0}, x^{(6)}_{9,0}, x^{(6)}_{13,0}$ can be expanded using Lemma 1 with probability 1. Next, we can combine the following factors: $x^{(6)}_{1,6}, x^{(6)}_{1,7}, x^{(6)}_{5,7}, x^{(6)}_{9,6}$ using Eq. (29) of Lemma 9 ($k = 3$); $x^{(6)}_{1,11}, x^{(6)}_{1,12}, x^{(6)}_{9,12}$ using Eq. (25) of Lemma 9 ($k = 4$); $x^{(6)}_{1,22}, x^{(6)}_{1,23}$ using Lemma 7 ($k = 3$); $x^{(6)}_{13,14}, x^{(6)}_{13,15}$ using Lemma 7 ($k = 1$); $x^{(6)}_{13,26}, x^{(6)}_{13,27}$ using Lemma 7 ($k = 1$). Finally, it remains some single terms to be expanded: $x^{(6)}_{9,26}$ using Lemma 3 ($k = 2$); $x^{(6)}_{13,24}$ using Lemma 3 ($k = 1$). By the Piling-up Lemma, we can combine these linear relations to obtain

$$x^{(6)}_1[0,6,7,11,12,22,23] \oplus x^{(6)}_5[7] \oplus x^{(6)}_9[0,6,12,26] \oplus x^{(6)}_{13}[0,14,15,24,26,27] = x^{(7)}_1[0,5,7,8,10,11,14,15,16,22,23,24,25,27,30,31] \oplus x^{(7)}_5[1,2,9,10,13,14,18,21,22,25,29,30] \oplus x^{(7)}_9[2,3,6,7,14,15,18,27] \oplus x^{(7)}_{13}[1,2,6,7,8,10,11,13,14,16,18,20,22,23,24,25,26] \tag{33}$$

with probability $\frac{1}{2}\left(1 + \frac{1}{2^{15}}\right)$.

In Group III, the factors $x^{(6)}_{2,0}$ and $x^{(6)}_{10,0}$ can be expanded using Lemma 1 with probability 1. Next, we can combine the following factors: $x^{(6)}_{2,6}, x^{(6)}_{2,7}$ using Lemma 7 ($k = 3$); $x^{(6)}_{6,13}, x^{(6)}_{6,14}$ using Lemma 7 ($k = 1$); $x^{(6)}_{14,25}, x^{(6)}_{14,26}$ using Lemma 7 ($k = 1$); $x^{(6)}_{2,18}, x^{(6)}_{2,19}, x^{(6)}_{6,19}$ using Eq. (26) of Lemma 9 ($k = 3$); $x^{(6)}_{2,8}, x^{(6)}_{6,7}, x^{(6)}_{14,8}$ using Eq. (27) of Lemma 9 ($k = 2$). Finally, it remains some single terms to be expanded: $x^{(6)}_{2,16}$ using Lemma 6 ($k = 3$); $x^{(6)}_{2,24}$ using Lemma 6 ($k = 3$). By the Piling-up Lemma, we can combine these linear relations to obtain

$$x^{(6)}_2[0,6,7,8,16,18,19,24] \oplus x^{(6)}_6[7,13,14,19] \oplus x^{(6)}_{10}[0] \oplus x^{(6)}_{14}[8,25,26] = x^{(7)}_2[6,7,9,10,16,18,19,25,26] \oplus x^{(7)}_6[2,3,5,7,10,11,13,14,19,22,23,27,30,31] \oplus x^{(7)}_{10}[0,3,4,8,12,13,14,18,20,27,28,30] \oplus x^{(7)}_{14}[0,6,13,14,15,16,23,24] \tag{34}$$

with probability $\frac{1}{2}\left(1 + \frac{1}{2^{16}}\right)$.

In Group IV, we can combine the following factors: $x^{(6)}_{7,14}, x^{(6)}_{7,15}$ using Lemma 7 ($k = 1$); $x^{(6)}_{7,6}, x^{(6)}_{7,7}, x^{(6)}_{11,6}, x^{(6)}_{11,7}$ using Eq. (28) of Lemma 9 ($k = 1$). It remains some single terms to be expanded: $x^{(6)}_{7,26}$ using Lemma 3 ($k = 1$); $x^{(6)}_{15,24}$ using Lemma 3 ($k = 1$). By the Piling-up Lemma, we can combine these linear relations to obtain

$$x^{(6)}_7[6,7,14,15,26] \oplus x^{(6)}_{11}[6,7] \oplus x^{(6)}_{15}[24] = x^{(7)}_3[6,7,8,24] \oplus x^{(7)}_7[1,2,13,25,26,30,31] \oplus x^{(7)}_{11}[6,14,15,18,19,23,24,27] \oplus x^{(7)}_{15}[16,25,26] \tag{35}$$

with probability $\frac{1}{2}\left(1 + \frac{1}{2^4}\right)$.

Finally, using the Piling-up Lemma we can combine the results from Lemma 10 and Eqs. (32)–(35), which leads to a correlation of $\varepsilon_L = 1/2^{8+12+15+16+4} = 2^{-55}$. □

Computational Result 2 *The linear approximation of Eq. (32) holds computationally with* $\varepsilon_{L_1} = 0.000301 \approx 2^{-11.70}$. *This correlation was verified using* 2^{42} *random samples.*

Computational Result 3 *The linear approximation of Eq. (33) holds computationally with* $\varepsilon_{L_2} = 0.000100 \approx 2^{-13.29}$. *This correlation was verified using* 2^{42} *random samples.*

Computational Result 4 *The linear approximation of Eq. (34) holds computationally with* $\varepsilon_{L_3} = 0.000051 \approx 2^{-14.26}$. *This correlation was verified using* 2^{42} *random samples.*

Computational Result 5 *The linear approximation of Eq. (35) holds computationally with* $\varepsilon_{L_4} = 0.0625 \approx 2^{-4}$. *This correlation was verified using* 2^{38} *random samples.*

Next, we will only work with a linear approximation for the bit $x_{5,0}^{(3.5)}$. As we will see in the next session, we are able to find a differential correlation to this bit (as introduced in [9], half a round of ChaCha consists in applying half the operations of the QRF. Thus, from Eq. (2) we can write $x_a^{(r-1/2)} = x_{a\prime}^{(r-1)}$, $\ldots$ $x_d^{(r-1/2)} = x_{d\prime}^{(r-1)}$). Using Eq. (5) it is easy to see that we have $x_{5,0}^{(3.5)} = x_{5,7}^{(4)} \oplus x_{10,0}^{(4)}$. Additionally, using Lemma 3 we can expand one more round and we get

$$x_{5,0}^{(3.5)} = x_{2,0}^{(5)} \oplus x_{5,26}^{(5)} \oplus x_{9,7}^{(5)} \oplus x_{9,19}^{(5)} \oplus x_{10,0}^{(5)} \oplus x_{13,6}^{(5)} \oplus x_{13,7}^{(5)} \oplus x_{14,0}^{(5)} \oplus x_{14,8}^{(5)}, \quad (36)$$

with probability $\frac{1}{2}\left(1 + \frac{1}{2}\right)$.

Lemma 12. *The following linear approximation holds with probability* $\frac{1}{2}\left(1 + \frac{1}{2^8}\right)$

$$\begin{aligned} x_{5,0}^{(3.5)} = {} & x_0^{(6)}[0] \oplus x_2^{(6)}[0,6,7,22,23] \oplus x_3^{(6)}[0,6,7,8,16,18,19,24] \oplus \\ & x_4^{(6)}[7,14,15] \oplus x_5^{(6)}[13] \oplus x_7^{(6)}[7,13,14,19] \oplus x_8^{(6)}[6,7,12] \oplus \\ & x_9^{(6)}[0,8,19] \oplus x_{10}^{(6)}[0,6,26] \oplus x_{13}^{(6)}[0,30,31] \oplus \\ & x_{14}^{(6)}[0,6,7,14,15,18,19,24,26,27] \oplus x_{15}^{(6)}[0,8,25,26]. \end{aligned}$$

Proof. We start from Eq. (36) and we can use Lemma 1 $x_{2,0}^{(5)}, x_{10,0}^{(5)}, x_{14,0}^{(5)}$ by $\mathcal{L}_{2,0}^{(6)}, \mathcal{L}_{10,0}^{(6)}, \mathcal{L}_{14,0}^{(6)}$ with probability 1. Next, note that, since we are transitioning from round 5 to 6, we have $(a,b,c,d) \in \{(0,5,10,15), (1,6,11,12), (2,7,8,13), (3,4,9,14)\}$. Considering $(a,b,c,d) = (0,5,10,15)$, we have the factor $x_{5,26}^{(5)}$ and we can apply Lemma 3 to get

$$\Pr\left(x_{5,26}^{(5)} = \mathcal{L}_{5,26}^{(6)} \oplus x_{15,25}^{(6)}\right) = \frac{1}{2}\left(1 + \frac{1}{2}\right).$$

Considering $(a,b,c,d) = (2,7,8,13)$, we have the factors $x_{13,6}^{(5)}$ and $x_{13,7}^{(5)}$. Then we can use Lemma 7 to get

$$\Pr\left(x_{13,6}^{(5)} \oplus x_{13,7}^{(5)} = \mathcal{L}_{13,6}^{(6)} \oplus \mathcal{L}_{13,7}^{(6)}\right) = \frac{1}{2}\left(1 + \frac{1}{2}\right).$$

Considering $(a, b, c, d) = (3, 4, 9, 14)$ we have $x^{(5)}_{9,7}, x^{(5)}_{9,19}$ and $x^{(5)}_{14,8}$, and then we can apply Lemma 3 to obtain

$$\Pr\left(x^{(5)}_{9,7} = \mathcal{L}^{(6)}_{9,7} \oplus x^{(6)}_{3,6} \oplus x^{(6)}_{14,14} \oplus x^{(6)}_{14,6}\right) = \frac{1}{2}\left(1 + \frac{1}{2^2}\right),$$

$$\Pr\left(x^{(5)}_{9,19} = \mathcal{L}^{(6)}_{9,19} \oplus x^{(6)}_{3,18} \oplus x^{(6)}_{14,26} \oplus x^{(6)}_{14,18}\right) = \frac{1}{2}\left(1 + \frac{1}{2^2}\right),$$

$$\Pr\left(x^{(5)}_{14,8} = \mathcal{L}^{(6)}_{14,8} \oplus x^{(6)}_{9,7} \oplus x^{(6)}_{4,14}\right) = \frac{1}{2}\left(1 + \frac{1}{2}\right).$$

By the Piling-up Lemma, we have that all these changes result in a probability of $\frac{1}{2}\left(1 + \frac{1}{2^8}\right)$. Expanding the linear terms using Eqs. (13)–(16) and canceling out common factors completes the proof. □

Computational Result 6 *The linear approximation of Lemma 12 holds computationally with* $\varepsilon_{L_0} = 0.00867 \approx 2^{-6.85}$.

Lemma 13. *The following linear approximation holds with probability* $\frac{1}{2}\left(1 + \frac{1}{2^{47}}\right)$

$$\begin{aligned} x^{(3.5)}_{5,0} = {} & x^{(7)}_0[0,6,7,11,12] \oplus x^{(7)}_1[7,8,14,15,16,18,19,30,31] \oplus \\ & x^{(7)}_2[0,2,3,5,6,8,10,11,14,15,16,18,19,24,25,27,30,31] \oplus \\ & x^{(7)}_3[6,7,9,10,18,19,25,26] \oplus x^{(7)}_4[1,2,7,19,26] \oplus x^{(7)}_5[0,5,6,7] \oplus \\ & x^{(7)}_6[1,2,9,10,19,21,22,29,31] \oplus x^{(7)}_7[2,3,5,10,11,13,14,19,22,23, \\ & 27,30,31] \oplus x^{(7)}_8[6,14,15,19,26,27] \oplus x^{(7)}_9[8,13,19,25,30,31] \oplus \\ & x^{(7)}_{10}[2,3,7,12,14,15,23,24,27] \oplus x^{(7)}_{11}[0,3,4,8,12,13,14,18,20,27, \\ & 28,30] \oplus x^{(7)}_{12}[0,5,6,11,12,19,20] \oplus x^{(7)}_{13}[0,7,12,13,15,16,18,19,22, \\ & 23,24,26,27] \oplus x^{(7)}_{14}[1,2,8,10,11,13,14,16,18,19,22,23,24,25,26, \\ & 30,31] \oplus x^{(7)}_{15}[5,6,7,8,13,14,15,16,23]. \end{aligned}$$

Proof. If we start from Lemma 12 we want to expand the equation one more round. To do so, first note that since we are transitioning from round 6 to 7, we have $(a, b, c, d) \in \{(0, 4, 8, 12), (1, 5, 9, 13), (2, 6, 10, 14), (3, 7, 11, 15)\}$. Therefore, we can divide the factors of the equation in 4 distinct groups:

- Group I - $x^{(6)}_0[0], x^{(6)}_4[7,14,15], x^{(6)}_8[6,7,12]$.
- Group II - $x^{(6)}_5[13], x^{(6)}_9[0,8,19], x^{(6)}_{13}[0,30,31]$.
- Group III - $x^{(6)}_2[0,6,7,22,23]$, $x^{(6)}_{10}[0,6,26]$, $x^{(6)}_{14}[0,6,7,14,15,18,19,24,26,27]$.
- Group IV - $x^{(6)}_3[0,6,7,8,16,18,19,24], x^{(6)}_7[7,13,14,19], x^{(6)}_{15}[0,8,25,26]$.

Here, we follow the same strategy as in the proof of Lemma 11. In Group I, the factor $x^{(6)}_{0,0}$ can be expanded using Lemma 1 with probability 1. Next, we can combine the following factors: $x^{(6)}_{4,7}, x^{(6)}_{8,6}, x^{(6)}_{8,7}$ using Lemma 8 ($k = 2$); $x^{(6)}_{4,14}, x^{(6)}_{4,15}$

using Lemma 7 with $(k = 1)$. Finally, we expand $x^{(6)}_{8,12}$ using Lemma 3 $(k = 2)$. By the Piling-up Lemma, we can combine these linear relations to obtain

$$\begin{aligned} &x^{(6)}_0[0] \oplus x^{(6)}_4[7, 14, 15] \oplus x^{(6)}_8[6, 7, 12] = x^{(7)}_0[0, 6, 7, 11, 12] \oplus \\ &x^{(7)}_4[1, 2, 7, 19, 26] \oplus x^{(7)}_8[6, 14, 15, 19, 26, 27] \oplus x^{(7)}_{12}[0, 5, 6, 11, 12, 19, 20] \end{aligned} \tag{37}$$

with probability $\frac{1}{2}\left(1 + \frac{1}{2^5}\right)$.

In Group II, the factors $x^{(6)}_{9,0}, x^{(6)}_{13,0}$ can be expanded using Lemma 1 with probability 1. Next, we can combine $x^{(6)}_{13,30}, x^{(6)}_{13,31}$ using Lemma 7 $(k = 1)$. The remaining terms can be expanded with Lemma 3: $x^{(6)}_{9,8}$ $(k = 2)$; $x^{(6)}_{9,19}$ $(k = 2)$; $x^{(6)}_{5,13}$ $(k = 1)$. By the Piling-up Lemma, we can combine these linear relations to obtain

$$\begin{aligned} &x^{(6)}_5[13] \oplus x^{(6)}_9[0, 8, 19] \oplus x^{(6)}_{13}[0, 30, 31] = x^{(7)}_1[7, 8, 14, 15, 16, 18, 19, 30, 31] \oplus \\ &x^{(7)}_5[0, 5, 6, 7] \oplus x^{(7)}_9[8, 13, 19, 25, 30, 31] \oplus x^{(7)}_{13}[0, 7, 12, 13, 15, 16, 18, 19, 22, \\ &23, 24, 26, 27] \end{aligned} \tag{38}$$

with probability $\frac{1}{2}\left(1 + \frac{1}{2^6}\right)$.

In Group III, the factors $x^{(6)}_{2,0}, x^{(6)}_{10,0}$ and $x^{(6)}_{14,0}$ can be expanded using Lemma 1 with probability 1. Next, we can combine the following factors: $x^{(6)}_{2,6}, x^{(6)}_{2,7}, x^{(6)}_{10,6}$, $x^{(6)}_{14,6}, x^{(6)}_{14,7}$ using Eq. (30) of Lemma 9 $(k = 3)$; $x^{(6)}_{2,22}, x^{(6)}_{2,23}$ using Lemma 7 $(k = 3)$; $x^{(6)}_{14,14}, x^{(6)}_{14,15}$ using Lemma 7 $(k = 1)$; $x^{(6)}_{14,18}, x^{(6)}_{14,19}$ using Lemma 7 $(k = 1)$; $x^{(6)}_{14,26}, x^{(6)}_{14,27}$ using Lemma 7 $(k = 1)$. Finally, it remains some single terms to be expanded: $x^{(6)}_{10,26}$ using Lemma 3 $(k = 2)$; $x^{(6)}_{14,24}$ using Lemma 6 $(k = 1)$. By the Piling-up Lemma, we can combine these linear relations to obtain

$$\begin{aligned} &x^{(6)}_2[0, 6, 7, 22, 23], x^{(6)}_{10}[0, 6, 26], x^{(6)}_{14}[0, 6, 7, 14, 15, 18, 19, 24, 26, 27] = \\ &x^{(7)}_2[0, 2, 3, 5, 6, 8, 10, 11, 14, 15, 16, 18, 19, 24, 25, 27, 30, 31] \oplus \\ &x^{(7)}_6[1, 2, 9, 10, 19, 21, 22, 29, 31] \oplus x^{(7)}_{10}[2, 3, 7, 12, 14, 15, 23, 24, 27] \oplus \\ &x^{(7)}_{14}[1, 2, 8, 10, 11, 13, 14, 16, 18, 19, 22, 23, 24, 25, 26, 30, 31] \end{aligned} \tag{39}$$

with probability $\frac{1}{2}\left(1 + \frac{1}{2^{12}}\right)$.

In Group IV, the factors $x^{(6)}_{3,0}$ and $x^{(6)}_{15,0}$ can be expanded using Lemma 1 with probability 1. Then we can combine the following factors: $x^{(6)}_{3,6}, x^{(6)}_{3,7}, x^{(6)}_{7,7}$ using Eq. (26) of Lemma 9 $(k = 3)$; $x^{(6)}_{3,18}, x^{(6)}_{3,19}, x^{(6)}_{7,19}$ using Eq. (26) of Lemma 9 $(k = 3)$; $x^{(6)}_{3,8}, x^{(6)}_{15,8}$ using Eq. (24) of Lemma 9 $(k = 2)$; $x^{(6)}_{15,25}, x^{(6)}_{15,26}$ using Lemma 7 $(k = 1)$; $x^{(6)}_{7,13}, x^{(6)}_{7,14}$ using Lemma 7 $(k = 1)$. It remains some single terms to be expanded: $x^{(6)}_{3,16}$ using Lemma 6 $(k = 3)$; $x^{(6)}_{3,24}$ using Lemma 6 $(k = 3)$. By the Piling-up Lemma, we can combine these linear relations to obtain

$$x_3^{(6)}[0,6,7,8,16,18,19,24], x_7^{(6)}[7,13,14,19], x_{15}^{(6)}[0,8,25,26] = x_3^{(7)}[6,7,9,10,18,19,25,26] \oplus x_7^{(7)}[2,3,5,10,11,13,14,19,22,23,27,30,31] \oplus x_{11}^{(7)}[0,3,4,8,12,13,14,18,20,27,28,30] \oplus x_{15}^{(7)}[5,6,7,8,13,14,15,16,23] \tag{40}$$

with probability $\frac{1}{2}\left(1+\frac{1}{2^{16}}\right)$.

Finally, using the Piling-up Lemma we can combine the results from Lemma 12 and Eqs. (37)-(40), which leads to a correlation of $\varepsilon_L = 1/2^{8+5+6+12+16} = 2^{-47}$. □

Computational Result 7 *The linear approximation of Eq. (37) holds computationally with* $\varepsilon_{L_1} = 0.0416 \approx 2^{-4.59}$. *This correlation was verified using* 2^{38} *random samples.*

Computational Result 8 *The linear approximation of Eq. (38) holds computationally with* $\varepsilon_{L_2} = 0.0278 \approx 2^{-5.19}$. *This correlation was verified using* 2^{38} *random samples.*

Computational Result 9 *The linear approximation of Eq. (39) holds computationally with* $\varepsilon_{L_3} = 0.000398 \approx 2^{-11.29}$. *This correlation was verified using* 2^{42} *random samples.*

Computational Result 10 *The linear approximation of Eq. (40) holds computationally with* $\varepsilon_{L_4} = 0.000047 \approx 2^{-14.38}$. *This correlation was verified using* 2^{42} *random samples.*

It is interesting to note that the experimental correlation is higher than expected in several cases. Of course, since the hypothesis of independence for the Piling-Up Lemma does not hold, it is expected to see deviations between what is predicted theoretically and what we see in practice. The fact that the correlation is usually higher indicates a positive correlation between some equations. In future works, it may be interesting to try to understand why ChaCha has this behavior.

4 Improved Differential-Linear Attacks Against ChaCha

4.1 New Differentials

In this section, we present new differentials for 3.5 rounds of ChaCha. As in previous works, these differential correlations were found experimentally. To find these correlations we used the technique proposed by Beierle et al. at Crypto 2020 [4], and described in Sect. 2.2. Here, the cipher is divided into the sub ciphers E_1 covering 1 round and E_2 covering 2.5 rounds to find a differential path for 3.5 rounds. Thus we want a particular differential characteristic of the form

$$\Delta X^{(0)} \xrightarrow{1 \text{ round}} \Delta X^{(1)} \xrightarrow{2.5 \text{ rounds}} \Delta X^{(3.5)}.$$

The idea is to generate consistent $\Delta X^{(1)}$ whose Hamming weight is minimized. In [4], the authors showed that the following differential characteristic occurs with probability 2^{-5} on average for the QRF of ChaCha

$$\begin{aligned}\Delta X^{(0)} = ((\![]), (\![]), (\![]), (\![i])) \rightarrow \Delta X^{(1)} = ((\![i+28]), (\![i+31, i+23, i+11,\\ i+3]), (\![i+24, i+16, i+4]),\\ (\![i+24, i+4])).\end{aligned} \tag{41}$$

From there we computed $\Delta X^{(3.5)}$ by generating random states $X^{(1)}$ and $X'^{(1)}$ and statistically testing for correlations in particular bits of $\Delta X^{(3.5)}$. We note that this procedure is computationally intensive as some of the correlations are very small. For some bits, we executed this procedure up to 2^{50} pairs of random states in the first round. To achieve this amount of computation we used 8 NVIDIA GPUs (RTX 2080ti). As in the referred paper, we used $i = 6$. Also, we fixed the differential of Eq. (41) in the third column of the state matrix. Table 3 shows the results.

Table 3. New differentials after 3.5 rounds, starting from $\Delta X^{(1)}$ in the third column of the state matrix with $i = 6$ in Eq. (41).

$\mathcal{OD}$	$\lvert\varepsilon_d\rvert$
$\Delta x_{0,0}^{(3.5)}$	0.000307
$\Delta x_{1,0}^{(3.5)}$	0.000124
$\Delta x_{12,0}^{(3.5)}$	0.000017
$\Delta x_{13,0}^{(3.5)}$	0.000016
$\Delta x_{5,0}^{(3.5)}$	0.0000002489

4.2 Distinguishers

Using the linear approximations of Lemma 10 and Lemma 11, the differential correlation $\varepsilon_d = 0.00048$ for $(a, b) = (3, 4)$ described in [10], and the estimated correlations from the Computational Results 1–5, we get $\varepsilon_d \varepsilon_{L_0}^2 \approx 2^{-25.37}$ which gives us a distinguisher for 6 rounds of ChaCha with complexity less than 2^{51}. Also, we get $\varepsilon_d(\varepsilon_{L_0}\varepsilon_{L_1}\varepsilon_{L_2}\varepsilon_{L_3}\varepsilon_{L_4})^2 \approx 2^{-111.86}$ which gives us a distinguisher for 7 rounds of ChaCha with complexity less than 2^{224}.

Using the linear approximations of Lemma 12 and Lemma 13, the differential correlation for $\Delta_{5,0}^{(3.5)}$ presented in Table 3, and the estimated correlations from the Computational Results 6–10, we get $\varepsilon_d \varepsilon_{L_0}^2 \approx 2^{-35.64}$ which gives us a distinguisher for 6 rounds of ChaCha with complexity less than $2^{72+5} = 2^{77}$ (here we have to repeat the procedure 2^5 times on average as in [4]). Also, we get $\varepsilon_d(\varepsilon_{L_0}\varepsilon_{L_1}\varepsilon_{L_2}\varepsilon_{L_3}\varepsilon_{L_4})^2 \approx 2^{-106.5}$ which gives us a distinguisher for 7 rounds of ChaCha with complexity less than $2^{213+5} = 2^{218}$.

4.3 New Attack Using Probabilistic Neutral Bits (PNBs)

One of the most important attacks against ChaCha is the proposal of Aumasson [1]. The attack first identifies good choices of truncated differentials, then it uses probabilistic backwards computation with the notion of PNBs, estimating the complexity of the attack. This attack is described in several previous works [1,22,23], thus, in our description, we skip several details.

The PNB-based key recovery is a fully experimental approach. We summarize the technique as follows:

- Let the correlation in the forward direction (a.k.a, differential-linear distinguisher) after r rounds be ε_d.
- Let n be the number of PNBs given by a correlation γ. Namely, even if we flip one bit in PNBs, we still observe correlation γ.
- Let the correlation in the backward direction, where all PNB bits are fixed to 0 and non-PNB bits are fixed to the correct ones, is ε_a

Then, the time complexity of the attack is estimated as $2^{256-n}N+2^{256-\alpha}$, where the data complexity N is given as

$$N = \left(\frac{\sqrt{\alpha \log(4)} + 3\sqrt{1-\varepsilon_a^2\varepsilon_d^2}}{\varepsilon_a\varepsilon_d} \right)^2,$$

where α is a parameter that the attacker can choose.

We can improve previous attacks for 7 rounds of ChaCha using this technique by considering the new differential correlation for $\Delta_{5,0}^{(3.5)}$ presented in Table 3. Using Eq. (5) it is easy to see that we have $x_{5,0}^{(3.5)} = x_{5,7}^{(4)} \oplus x_{10,0}^{(4)}$. Therefore, we consider $\mathcal{ID}$ given by Eq. (41) with $i = 6$ and $\mathcal{OD}$ $x_{5,7}^{(4)} \oplus x_{10,0}^{(4)}$. Using $\gamma = 0.35$ we found 108 PNBs, and we obtained $\varepsilon_a = 0.000169$. From that, we get an attack with data complexity of $2^{75.51}$ and time complexity $2^{223.51}$. As in [4], we have to repeat this attack 2^5 times on average. Thus, the final attack has data complexity of $2^{80.51}$ and time complexity $2^{228.51}$. Bellow we list all PNBs:

PNB = (2, 3, 4, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 19, 20, 21, 22, 23, 26, 27, 28, 29, 30, 31, 32, 33, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 66, 67, 68, 69, 78, 79, 80, 102, 103, 111, 112, 115, 128, 129, 130, 135, 136, 143, 144, 147, 148, 149, 150, 151, 155, 156, 157, 158, 159, 160, 161, 162, 163, 168, 169, 170, 173, 174, 175, 176, 179, 180, 181, 182, 185, 186, 191, 199, 200, 201, 219, 220, 221, 222, 223, 232, 255).

5 Conclusion

In this paper, we presented a new technique to find linear approximations for ARX ciphers. Applying this technique we presented new linear approximations to the stream cipher ChaCha which gave us new and improved distinguishers. In addition, we presented new differential characteristics for 3.5 rounds of ChaCha

and use them to improve the attacks based on Probabilistic Neutral Bits. For future works, we expect that the proposed technique can be used to improve attacks against similar ARX-based designs, as the stream cipher Salsa and the hash function Blake.

Acknowledgements. The authors would like to thank the anonymous reviewers for their valuable comments and suggestions which helped us to improve our work.

A Proofs

In this appendix, we expand the proof of Lemma 9 for each individual linear approximation.

A.1 Eq. (19)

Proof. Using Eqs. (9) and (10) we can write

$$x_{b,i}^{(m-1)} \oplus x_{c,i}^{(m-1)} = \mathcal{L}_{b,i}^{(m)} \oplus \Theta_i(x_c'^{(m-1)}, x_d^{(m)}) \oplus \mathcal{L}_{c,i}^{(m)} \oplus \Theta_i(x_c'^{(m-1)}, x_d^{(m)}) \oplus \Theta_i(x_c^{(m-1)}, x_d'^{(m-1)}).$$

Using the approximation of Eq. (17) we can write $\Theta_i(x_c^{(m-1)}, x_d'^{(m-1)}) = x_{d,i-1}'^{(m-1)}$ with probability $\frac{1}{2}\left(1+\frac{1}{2}\right)$. Thus, using Eq. (7) and canceling out common factors we get

$$x_{b,i}^{(m-1)} \oplus x_{c,i}^{(m-1)} = \mathcal{L}_{b,i}^{(m)} \oplus \mathcal{L}_{c,i}^{(m)} \oplus x_{a,i-1}^{(m)} \oplus x_{d,i+7}^{(m)},$$

with probability $\frac{1}{2}\left(1+\frac{1}{2}\right)$, which concludes the proof. □

A.2 Eqs. (20) and (21)

Proof. Using Eqs. (9) and (12) we can write

$$x_{a,i}^{(m-1)} \oplus x_{b,i}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{b,i}^{(m)} \oplus \Theta_i(x_c'^{(m-1)}, x_d^{(m)}) \oplus \Theta_i(x_c'^{(m-1)}, x_d^{(m)}) \oplus \Theta_i(x_a'^{(m-1)}, x_b'^{(m-1)}) \oplus \Theta_i(x_a^{(m-1)}, x_b^{(m-1)}).$$

Cancelling out common factors, using the approximation of Eq. (17) and the Piling-up Lemma we can write

$$x_{a,i}^{(m-1)} \oplus x_{b,i}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{b,i}^{(m)} \oplus x_{b,i-1}'^{(m-1)} \oplus x_{b,i-1}^{(m-1)}$$

with probability $\frac{1}{2}\left(1+\frac{1}{2^2}\right)$. Now we can replace $x_{b,i-1}'^{(m-1)}$ using Eq. (5) and $x_{b,i-1}^{(m-1)}$ using Lemma 3, which leads to

$$x_{a,i}^{(m-1)} \oplus x_{b,i}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{b,i}^{(m)} \oplus x_{b,i+6}^{(m)} \oplus x_{c,i-1}^{(m)} \oplus \mathcal{L}_{b,i-1}^{(m)} \oplus x_{d,i-2}^{(m)},$$

with probability $\frac{1}{2}\left(1+\frac{1}{2^3}\right)$ by the Piling-up Lemma. We can also use Lemma 1 in order to obtain

$$x_{a,1}^{(m-1)} \oplus x_{b,1}^{(m-1)} = \mathcal{L}_{a,1}^{(m)} \oplus \mathcal{L}_{b,1}^{(m)} \oplus x_{b,7}^{(m)} \oplus x_{c,0}^{(m)} \oplus \mathcal{L}_{b,0}^{(m)},$$

with probability $\frac{1}{2}\left(1+\frac{1}{2^2}\right)$. □

A.3 Eqs. (22) and (23)

Proof. Combining Eq. (10) and Eq. (12) we have

$$x_{a,i}^{(m-1)} \oplus x_{c,i}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{c,i}^{(m)} \oplus \Theta_i(x_c^{(m-1)}, x_d'^{(m-1)}) \oplus \Theta_i(x_a'^{(m-1)}, x_b'^{(m-1)}) \oplus \Theta_i(x_a^{(m-1)}, x_b^{(m-1)}).$$

Using the approximation of Eq. (17) and the Piling-up Lemma we can write

$$x_{a,i}^{(m-1)} \oplus x_{c,i}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{c,i}^{(m)} \oplus x_{d,i-1}'^{(m-1)} \oplus x_{b,i-1}'^{(m-1)} \oplus x_{b,i-1}^{(m-1)}$$

with probability $\frac{1}{2}\left(1 + \frac{1}{2^3}\right)$. Now we can replace $x_{d,i-1}'^{(m-1)}$ using Eq. (7), $x_{b,i-1}'^{(m-1)}$ using Eq. (5) and $x_{b,i-1}^{(m-1)}$ using Lemma 3 if $i > 1$ or 1 if $i = 1$, which leads to

$$x_{a,i}^{(m-1)} \oplus x_{c,i}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{c,i}^{(m)} \oplus x_{a,i-1}^{(m)} \oplus x_{d,i+7}^{(m)} \oplus x_{b,i+6}^{(m)} \oplus x_{c,i-1}^{(m)} \oplus \mathcal{L}_{b,i-1}^{(m)} \oplus x_{d,i-2}^{(m)},$$

with probability $\frac{1}{2}\left(1 + \frac{1}{2^4}\right)$ by the Piling-up Lemma or

$$x_{a,1}^{(m-1)} \oplus x_{c,1}^{(m-1)} = \mathcal{L}_{a,1}^{(m)} \oplus \mathcal{L}_{c,1}^{(m)} \oplus x_{a,0}^{(m)} \oplus x_{d,8}^{(m)} \oplus x_{b,7}^{(m)} \oplus x_{c,0}^{(m)} \oplus \mathcal{L}_{b,0}^{(m)},$$

with probability $\frac{1}{2}\left(1 + \frac{1}{2^3}\right)$. □

A.4 Eq. (24)

Proof. Using Eq. (11) and Eq. (12) we can write

$$x_{a,i}^{(m-1)} \oplus x_{d,i}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{d,i}^{(m)} \oplus \Theta_i(x_c'^{(m-1)}, x_d^{(m)}) \oplus \Theta_i(x_a^{(m-1)}, x_b^{(m-1)}).$$

Using Eq. (17) we get

$$x_{a,i}^{(m-1)} \oplus x_{d,i}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{d,i}^{(m)} \oplus \Theta_i(x_c'^{(m-1)}, x_d^{(m)}) \oplus x_{b,i-1}^{(m-1)},$$

and from Eq. (9)

$$x_{a,i}^{(m-1)} \oplus x_{d,i}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{d,i}^{(m)} \oplus \Theta_i(x_c'^{(m-1)}, x_d^{(m)}) \oplus \mathcal{L}_{b,i-1}^{(m)} \oplus \Theta_{i-1}(x_c'^{(m-1)}, x_d^{(m)}),$$

with probability $\frac{1}{2}\left(1 + \frac{1}{2}\right)$. Thus, using the approximation of Eq. (18) and the Piling-up Lemma we can write

$$x_{a,i}^{(m-1)} \oplus x_{d,i}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{d,i}^{(m)} \oplus \mathcal{L}_{b,i-1}^{(m)},$$

with probability $\frac{1}{2}\left(1 + \frac{1}{2^2}\right)$. □

A.5 Eq. (25)

Proof. Using Eq. (12) and Eq. (10) and canceling out common factors we get

$$\begin{aligned} x_{a,i-1}^{(m-1)} \oplus x_{a,i}^{(m-1)} \oplus x_{c,i}^{(m-1)} = {} & \mathcal{L}_{a,i-1}^{(m)} \oplus \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{c,i}^{(m)} \oplus \\ & \Theta_{i-1}(x_a'^{(m-1)}, x_b'^{(m-1)}) \oplus \Theta_{i-1}(x_c'^{(m-1)}, x_d^{(m)}) \oplus \\ & \Theta_{i-1}(x_a^{(m-1)}, x_b^{(m-1)}) \oplus \Theta_i(x_a'^{(m-1)}, x_b'^{(m-1)}) \oplus \\ & \Theta_i(x_a^{(m-1)}, x_b^{(m-1)}) \oplus \Theta_i(x_c^{(m-1)}, x_d'^{(m-1)}) \end{aligned}$$

Using the approximation of Eq. (18) and the Piling-up Lemma we obtain

$$\begin{aligned} x_{a,i-1}^{(m-1)} \oplus x_{a,i}^{(m-1)} \oplus x_{c,i}^{(m-1)} = {} & \mathcal{L}_{a,i-1}^{(m)} \oplus \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{c,i}^{(m)} \oplus \\ & \Theta_{i-1}(x_c'^{(m-1)}, x_d^{(m)}) \oplus \Theta_i(x_c^{(m-1)}, x_d'^{(m-1)}) \end{aligned}$$

with probability $\frac{1}{2}\left(1 + \frac{1}{2^2}\right)$. Using Eq. (17) and Eq. (7) we get

$$\begin{aligned} x_{a,i-1}^{(m-1)} \oplus x_{a,i}^{(m-1)} \oplus x_{c,i}^{(m-1)} = {} & \mathcal{L}_{a,i-1}^{(m)} \oplus \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{c,i}^{(m)} \oplus \\ & x_{d,i-2}^{(m)} \oplus x_{a,i-1}^{(m)} \oplus x_{d,i+7}^{(m)} \end{aligned}$$

with probability $\frac{1}{2}\left(1 + \frac{1}{2^4}\right)$. □

A.6 Eq. (26)

Proof. Using Eq. (9) and Eq. (12) and canceling out common factors we can write

$$\begin{aligned} & x_{a,i}^{(m-1)} \oplus x_{a,i-1}^{(m-1)} \oplus x_{b,i}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{a,i-1}^{(m)} \oplus \mathcal{L}_{b,i}^{(m)} \oplus \\ & \Theta_{i-1}(x_a'^{(m-1)}, x_b'^{(m-1)}) \oplus \Theta_{i-1}(x_c'^{(m-1)}, x_d^{(m)}) \oplus \Theta_{i-1}(x_a^{(m-1)}, x_b^{(m-1)}) \oplus \\ & \Theta_i(x_a'^{(m-1)}, x_b'^{(m-1)}) \oplus \Theta_i(x_a^{(m-1)}, x_b^{(m-1)}). \end{aligned}$$

Using the approximation of Eq. (18) and the Piling-up Lemma we can write

$$\begin{aligned} x_{a,i}^{(m-1)} \oplus x_{a,i-1}^{(m-1)} \oplus x_{b,i}^{(m-1)} = {} & \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{a,i-1}^{(m)} \\ & \oplus \mathcal{L}_{b,i}^{(m)} \oplus \Theta_{i-1}(x_c'^{(m-1)}, x_d^{(m)}). \end{aligned}$$

with probability $\frac{1}{2}\left(1 + \frac{1}{2^2}\right)$. Using the approximation of Eq. (17) we get

$$x_{a,i}^{(m-1)} \oplus x_{a,i-1}^{(m-1)} \oplus x_{b,i}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{a,i-1}^{(m)} \oplus \mathcal{L}_{b,i}^{(m)} \oplus x_{d,i-2}^{(m)}.$$

with probability $\frac{1}{2}\left(1 + \frac{1}{2^3}\right)$. □

A.7 Eq. (27)

Proof. Using Eq. (11) and Eq. (12), and canceling out common factors we have

$$x_{b,i-1}^{(m-1)} \oplus x_{a,i}^{(m-1)} \oplus x_{d,i}^{(m-1)} = x_{b,i-1}^{(m-1)} \oplus \mathcal{L}_{a,i}^{(m)} \oplus \Theta_i(x_c'^{(m-1)}, x_d^{(m)}) \oplus \Theta_i(x_a^{(m-1)}, x_b^{(m-1)}) \oplus \mathcal{L}_{d,i}^{(m)}.$$

Using the approximation of Eq. (17) we have $\Theta_i(x_a^{(m-1)}, x_b^{(m-1)}) = x_{b,i-1}^{(m-1)}$ occurring with probability $\frac{1}{2}\left(1 + \frac{1}{2^2}\right)$. Then

$$x_{b,i-1}^{(m-1)} \oplus x_{a,i}^{(m-1)} \oplus x_{d,i}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{d,i}^{(m)} \oplus \Theta_i(x_c'^{(m-1)}, x_d^{(m)}).$$

with probability $\frac{1}{2}\left(1 + \frac{1}{2}\right)$. Finally, using the approximation of Eq. (17) and the Piling-up Lemma we get

$$x_{b,i-1}^{(m-1)} \oplus x_{a,i}^{(m-1)} \oplus x_{d,i}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{d,i}^{(m)} \oplus x_{d,i-1}^{(m)}.$$

with probability $\frac{1}{2}\left(1 + \frac{1}{2^2}\right)$. □

A.8 Eq. (28)

Proof. Using Eq. (9) and Eq. (10), we can write

$$x_{b,i-1}^{(m-1)} \oplus x_{b,i}^{(m-1)} \oplus x_{c,i-1}^{(m-1)} \oplus x_{c,i}^{(m-1)} = \mathcal{L}_{b,i-1}^{(m)} \oplus \Theta_{i-1}(x_c'^{(m-1)}, x_d^{(m)}) \oplus \mathcal{L}_{b,i}^{(m)} \oplus \Theta_i(x_c'^{(m-1)}, x_d^{(m)}) \oplus \mathcal{L}_{c,i-1}^{(m)} \oplus \Theta_{i-1}(x_c'^{(m-1)}, x_d^{(m)}) \oplus \Theta_{i-1}(x_c^{(m-1)}, x_d'^{(m-1)}) \oplus \mathcal{L}_{c,i}^{(m)} \oplus \Theta_i(x_c'^{(m-1)}, x_d^{(m)}) \oplus \Theta_i(x_c^{(m-1)}, x_d'^{(m-1)}).$$

Canceling out common factors we get

$$x_{b,i-1}^{(m-1)} \oplus x_{b,i}^{(m-1)} \oplus x_{c,i-1}^{(m-1)} \oplus x_{c,i}^{(m-1)} = \mathcal{L}_{b,i-1}^{(m)} \oplus \mathcal{L}_{b,i}^{(m)} \oplus \mathcal{L}_{c,i-1}^{(m)} \oplus \mathcal{L}_{c,i}^{(m)} \oplus \Theta_{i-1}(x_c^{(m-1)}, x_d'^{(m-1)}) \oplus \Theta_i(x_c^{(m-1)}, x_d'^{(m-1)}).$$

Thus, using the approximation of Eq. (18) we get

$$x_{b,i-1}^{(m-1)} \oplus x_{b,i}^{(m-1)} \oplus x_{c,i-1}^{(m-1)} \oplus x_{c,i}^{(m-1)} = \mathcal{L}_{b,i-1}^{(m)} \oplus \mathcal{L}_{b,i}^{(m)} \oplus \mathcal{L}_{c,i-1}^{(m)} \oplus \mathcal{L}_{c,i}^{(m)}.$$

with probability $\frac{1}{2}\left(1 + \frac{1}{2}\right)$. □

A.9 Eq. (29)

Proof. Using Eqs. (9), (10) and (12)

$$x_{a,i}^{(m-1)} \oplus x_{a,i-1}^{(m-1)} \oplus x_{b,i}^{(m-1)} \oplus x_{c,i-1}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{a,i-1}^{(m)} \oplus \mathcal{L}_{b,i}^{(m)} \oplus \mathcal{L}_{c,i-1}^{(m)} \oplus \Theta_i(x_a'^{(m-1)}, x_b'^{(m-1)}) \oplus \Theta_i(x_a^{(m-1)}, x_b^{(m-1)}) \oplus \Theta_{i-1}(x_a'^{(m-1)}, x_b'^{(m-1)}) \oplus \Theta_{i-1}(x_a^{(m-1)}, x_b^{(m-1)}) \oplus \Theta_{i-1}(x_c^{(m-1)}, x_d'^{(m-1)}).$$

Using the approximation of Eq. (18) and the Piling-up Lemma we can write

$$x_{a,i}^{(m-1)} \oplus x_{a,i-1}^{(m-1)} \oplus x_{b,i}^{(m-1)} \oplus x_{c,i-1}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{a,i-1}^{(m)} \oplus \mathcal{L}_{b,i}^{(m)} \oplus \mathcal{L}_{c,i-1}^{(m)} \oplus \Theta_{i-1}(x_c^{(m-1)}, x_d'^{(m-1)}).$$

with probability $\frac{1}{2}\left(1 + \frac{1}{2^2}\right)$. Therefore, Eqs. (17) and (7) give us

$$x_{a,i}^{(m-1)} \oplus x_{a,i-1}^{(m-1)} \oplus x_{b,i}^{(m-1)} \oplus x_{c,i-1}^{(m-1)} = \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{a,i-1}^{(m)} \oplus \mathcal{L}_{b,i}^{(m)} \oplus \mathcal{L}_{c,i-1}^{(m)} \oplus x_{a,i-2}^{(m)} \oplus x_{d,i+6}^{(m)}.$$

with probability $\frac{1}{2}\left(1 + \frac{1}{2^3}\right)$. □

A.10 Eq. (30)

Proof. Using Eqs. (10), (11) and (12), we can write

$$x_{a,i}^{(m-1)} \oplus x_{a,i-1}^{(m-1)} \oplus x_{c,i-1}^{(m-1)} \oplus x_{d,i}^{(m-1)} \oplus x_{d,i-1}^{(m-1)} = \mathcal{L}_{a,i-1}^{(m)} \oplus \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{c,i-1}^{(m)} \oplus \mathcal{L}_{d,i-1}^{(m)} \oplus \mathcal{L}_{d,i}^{(m)} \oplus \Theta_{i-1}(x_a^{(m-1)}, x_b^{(m-1)}) \oplus \Theta_i(x_c'^{(m-1)}, x_d^{(m)}) \oplus \Theta_i(x_a^{(m-1)}, x_b^{(m-1)}) \oplus \Theta_{i-1}(x_c^{(m-1)}, x_d'^{(m-1)}).$$

Using the approximation of Eq. (18) we have

$$x_{a,i}^{(m-1)} \oplus x_{a,i-1}^{(m-1)} \oplus x_{c,i-1}^{(m-1)} \oplus x_{d,i}^{(m-1)} \oplus x_{d,i-1}^{(m-1)} = \mathcal{L}_{a,i-1}^{(m)} \oplus \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{c,i-1}^{(m)} \oplus \mathcal{L}_{d,i-1}^{(m)} \oplus \mathcal{L}_{d,i}^{(m)} \oplus \Theta_i(x_c'^{(m-1)}, x_d^{(m)}) \oplus \Theta_{i-1}(x_c^{(m-1)}, x_d'^{(m-1)})$$

with probability $\frac{1}{2}\left(1 + \frac{1}{2}\right)$. Finally, by the Piling-up Lemma and using the approximation of Eq. (17) and Eq. (7), we get

$$x_{a,i}^{(m-1)} \oplus x_{a,i-1}^{(m-1)} \oplus x_{c,i-1}^{(m-1)} \oplus x_{d,i}^{(m-1)} \oplus x_{d,i-1}^{(m-1)} = \mathcal{L}_{a,i-1}^{(m)} \oplus \mathcal{L}_{a,i}^{(m)} \oplus \mathcal{L}_{c,i-1}^{(m)} \oplus \mathcal{L}_{d,i-1}^{(m)} \oplus \mathcal{L}_{d,i}^{(m)} \oplus x_{d,i-1}^{(m)} \oplus x_{a,i-2}^{(m)} \oplus x_{d,i+6}^{(m)}$$

with probability $\frac{1}{2}\left(1 + \frac{1}{2^3}\right)$. □

References

1. Aumasson, J.-P., Fischer, S., Khazaei, S., Meier, W., Rechberger, C.: New features of Latin dances: analysis of Salsa, ChaCha, and Rumba. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 470–488. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-71039-4_30
2. Aumasson, J.P., Henzen, L., Meier, W., Phan, R.C.W.: SHA-3 proposal blake. Submission to NIST 92 (2008)
3. Beierle, C., et al.: Schwaemm and Esch: lightweight authenticated encryption and hashing using the Sparkle permutation family (2019)
4. Beierle, C., Leander, G., Todo, Y.: Improved differential-linear attacks with applications to ARX ciphers. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12172, pp. 329–358. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56877-1_12

5. Bernstein, D.J.: The poly1305-AES message-authentication code. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 32–49. Springer, Heidelberg (2005). https://doi.org/10.1007/11502760_3
6. Bernstein, D.J.: ChaCha, a variant of Salsa20. In: Workshop Record of SASC, vol. 8, 3–5 (2008)
7. Bernstein, D.J.: The Salsa20 family of stream ciphers. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 84–97. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68351-3_8
8. Blondeau, C., Leander, G., Nyberg, K.: Differential-linear cryptanalysis revisited. J. Cryptol. **30**(3), 859–888 (2016). https://doi.org/10.1007/s00145-016-9237-5
9. Choudhuri, A.R., Maitra, S.: Significantly improved multi-bit differentials for reduced round Salsa and Chacha. IACR Transa. Symmetric Cryptol. 261–287 (2016)
10. Coutinho, M., Neto, T.S.: New multi-bit differentials to improve attacks against ChaCha. IACR Cryptology ePrint Archive 2020, 350 (2020)
11. Crowley, P.: Truncated differential cryptanalysis of five rounds of Salsa20. In: The State of the Art of Stream Ciphers SASC 2006, pp. 198–202 (2006)
12. Dey, S., Roy, T., Sarkar, S.: Revisiting design principles of Salsa and ChaCha. Adv. Math. Commun. **13**(4), 689 (2019)
13. Dey, S., Sarkar, S.: Improved analysis for reduced round Salsa and Chacha. Discrete Appl. Math. **227**, 58–69 (2017)
14. Ding, L.: Improved related-cipher attack on Salsa20 stream cipher. IEEE Access **7**, 30197–30202 (2019)
15. Dinu, D., Perrin, L., Udovenko, A., Velichkov, V., Großschädl, J., Biryukov, A.: Design strategies for ARX with provable bounds: Sparx and LAX. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 484–513. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_18
16. Fischer, S., Meier, W., Berbain, C., Biasse, J.-F., Robshaw, M.J.B.: Non-randomness in eSTREAM Candidates Salsa20 and TSC-4. In: Barua, R., Lange, T. (eds.) INDOCRYPT 2006. LNCS, vol. 4329, pp. 2–16. Springer, Heidelberg (2006). https://doi.org/10.1007/11941378_2
17. Hernandez-Castro, J.C., Tapiador, J.M.E., Quisquater, J.-J.: On the Salsa20 core function. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 462–469. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-71039-4_29
18. IANIX: ChaCha usage & deployment (2020). https://ianix.com/pub/chacha-deployment.html. Accessed 13 Jan 2020
19. Ishiguro, T., Kiyomoto, S., Miyake, Y.: Latin dances revisited: new analytic results of Salsa20 and ChaCha. In: Qing, S., Susilo, W., Wang, G., Liu, D. (eds.) ICICS 2011. LNCS, vol. 7043, pp. 255–266. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25243-3_21
20. Langford, S.K., Hellman, M.E.: Differential-linear cryptanalysis. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 17–25. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48658-5_3
21. Langley, A., Chang, W., Mavrogiannopoulos, N., Strombergson, J., Josefsson, S.: ChaCha20-Poly1305 cipher suites for transport layer security (TLS). RFC 7905 (10) (2016)
22. Maitra, S., Paul, G., Meier, W.: Salsa20 cryptanalysis: new moves and revisiting old styles. In: The Ninth International Workshop on Coding and Cryptography (2015)
23. Maitra, S.: Chosen IV cryptanalysis on reduced round ChaCha and Salsa. Discrete Appl. Math. **208**, 88–97 (2016)

24. Mouha, N., Preneel, B.: A proof that the ARX cipher Salsa20 is secure against differential cryptanalysis. IACR Cryptology ePrint Archive 2013, 328 (2013)
25. Muller, S.: Documentation and analysis of the Linux random number generator - federal office for information security (Germany's) (2019). https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Studies/LinuxRNG/LinuxRNG_EN.pdf;jsessionid=6B0F8D7795B80F5EADA3DB3DB3E4043B.1_cid360?__blob=publicationFile&v=19
26. Robshaw, M., Billet, O. (eds.): New Stream Cipher Designs. LNCS, vol. 4986. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68351-3
27. Shi, Z., Zhang, B., Feng, D., Wu, W.: Improved key recovery attacks on reduced-round Salsa20 and ChaCha. In: Kwon, T., Lee, M.-K., Kwon, D. (eds.) ICISC 2012. LNCS, vol. 7839, pp. 337–351. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37682-5_24
28. Torvalds, L.: Linux kernel source tree (2016). https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=818e607b57c94ade9824dad63a96c2ea6b21baf3
29. Tsunoo, Y., Saito, T., Kubo, H., Suzaki, T., Nakashima, H.: Differential cryptanalysis of Salsa20/8. In: Workshop Record of SASC, vol. 28 (2007)
30. Wallén, J.: Linear approximations of addition modulo 2^n. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 261–273. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-39887-5_20

Artigos publicados em 2022

libharpia: a New Cryptographic Library for Brazilian Elections

Rodrigo Pacheco[1], Douglas Braga[1], Iago Passos[1], Thiago Araújo[1], Vinícius Lagrota[1], Murilo Coutinho[1]

[1]Research and Development Center for Communication Security (CEPESC)

Abstract. *The Research and Development Center for Communication Security (CEPESC) has a long partnership history with the Brazilian Superior Electoral Court to improve the security of the Brazilian election system. Among all the contributions from CEPESC, probably the most important is a cryptographic library used in some critical moments during the election. In an effort to improve transparency and auditability of the solution, we present the new cryptographic library developed at CEPESC, named* `libharpia`*. Its main design goal is to allow transparency and readability while substantially increasing security. One of the main advances is the use of post-quantum cryptography, implemented through secure hybrid protocols that mix current cryptographic standards (specifically elliptic curves) with new cryptographic primitives based on Lattices, believed to be secure against quantum computers.*

1. Introduction

The Brazilian electronic voting machine, or simply "Urna Eletrônica" (UE), was developed by the Brazilian Superior Electoral Court (Tribunal Superior Eleitoral - TSE) and is part of the Brazilian electoral process since 1996. Along these years we have seen great progress in the system and in the process itself. In the beginning, the UE was based on personal computers, implementing a VirtuOS and Windows CE, and was designed by third parties. From 2005 onwards, the development of the most critical parts of the systems has been developed by TSE's engineers, including a new operating system based on Linux deployed in 2008 [Monteiro et al. 2019]. In 2009, the security of the system was greatly improved by the adoption of a hardware trusted computing base (TCB) for direct recording electronic voting architecture, T-DRE in short. This device is capable of performing several important cryptographic functionalities such as key storage, random number generation, digital signatures and the authentication of critical softwares (e.g., BIOS, bootloader, Linux kernel) [Gallo et al. 2010]. Another great advance was the use of biometric data to uniquely identify the voters, a process that started in 2008 and now reach over 80% of the electorate. More recently, TSE announced that new voting machines are going to use digital certificates from Brazilian national PKI (ICP-Brasil) [TSE 2021].

All these initiatives showcase TSE's efforts into making Brazilian elections secure and trustworthy. Nevertheless, from the beginning, the Brazilian voting system has been criticized in terms of its transparency and auditability. For example, in 2002, a report from the Brazilian Computer Society (SBC) presented a complete analysis of the election system. Although no signs of fraud or big security problems were found, the authors appointed a lack transparency and auditability as

a concern [van de Graaf and Custódio 2002]. Another report with similar conclusions [Brunazo Filho et al. 2015] was published after a very tight election in 2014.

Along the years, TSE has been working to improve these limitations. Starting in 2009, one important step in this direction was the organization of several hacking challenges, the so called "Teste Público de Segurança" (TPS), where external and independent researchers can examine the security mechanisms implemented within the system, find vulnerabilities and provide suggestions for improvement (see [Aranha et al. 2019] for a review of the problems detected in these tests). We note that the vulnerabilities found are always patched by TSE and the fixes presented to researchers. In addition to the TPS, TSE has progressively been opening the system by open-sourcing parts of the solution [Alessandre 2021] and through academic publications [Coimbra et al. 2017, Monteiro et al. 2019].

However, there is still room for improvement. For example, one common criticism is the participation of the Research and Development Center for Communication Security (CEPESC) in providing cryptographic libraries for TSE. Indeed, a report published in 2015 said that "TSE accepts in the electoral system the existence of code and services of authentication provided by CEPESC, constituting a critical point of failure that can influence almost all routines in the electoral system" [Brunazo Filho et al. 2015]. The criticism was valid, since up to that point, the source code developed by CEPESC was not known by the community and was not a part of the TPS.

Created in 1982, CEPESC has four decades of experience, research and development in cryptology, there is no other institution in Brazil with such an experience in the area. CEPESC's primary mission is to develop secure solutions to protect the most critical information of the Brazilian government. CEPESC has a long partnership history with TSE, being an important part of the electronic election system since its inception in 1996. During this period, CEPESC helped TSE in several critical activities, such as: (i) evaluating the security of critical software and hardware components developed by third parties; (ii) testing the quality of the random number generators of the UE and its peripherals; (iii) helping TSE to develop protocols and evaluating operational security; (iv) developing cryptographic libraries and algorithms.

Traditionally, CEPESC's solutions were treated as proprietary and their source code were never disclosed to the general public. This reality is changing in order to improve transparency and auditability of the election system, both key aspects of democracy. This work is a further step in this direction, as the organizational agreement with TSE already establishes the necessity of opening the design choices and possibly the source code itself. Therefore, in this work, we describe the new version of the cryptographic library developed by CEPESC, called `libharpia`. One of the goals of this work is to achieve a higher social trust in the primitives and protocols implemented.

The main advantage of `libharpia` is the use of post-quantum cryptography implemented through secure hybrid protocols that mix current standards (based on elliptic curves) with new cryptographic primitives (based on lattices). More precisely, we use the algorithms Kyber and Dilithium [Ducas et al. 2018, Bos et al. 2018] to deliver a Key Encapsulation Mechanism (KEM) and digital signatures, respectively. These algorithms were recently announced among the winners in the post-quantum cryptography standard-

ization process organized by NIST [NIST 2022]. To the best of our knowledge, this will make Brazilian elections the first in the world using post-quantum cryptography and the first system in general using this kind of cryptography in any Brazilian public institution.

This work is organized as follows: in Section 2, we provide a description of the goals and scope of `libharpia`. In Sections 3 and 4, we present the main choices in terms of implementation and cryptographic primitives, respectively. Then, in Section 5 we present the API of `libharpia`. Also, in Section 6, we go into more details over the cryptographic protocols implemented, providing justification for their choices and security. Finally, in Section 7 we present the conclusions.

2. Goals and Scope

`libharpia` is used by TSE in the following situations:

1. The TSE prepares over 500 000 UE machines for the election. To do so, a media containing all software and data necessary is created and used to inject them into the UE. To make sure that an adversary cannot create a fake media, all binaries are signed using `libharpia` and TSE's private key to prove their authenticity.
2. During the election, voters are authenticated using their biometric information. To make sure that this private information is protected, `libharpia` is used to encrypt all the data.
3. When all voters from a particular section finish voting, the UE emits a ballot box ("Boletim de Urna"), that is a summary of the results of a specific machine. Then, the UE encrypts and signs the ballot box using `libharpia`. This step protects the ballot box from tampering during transmission.
4. TSE's tally server receives the encrypted ballot box from every single UE machine across the country. Then, it verifies the digital signature and decrypts the ballot box to access the results and add them to conclude the election.

It is important to note that `libharpia` is not the only means of cryptographic protection in the electoral system. In fact, TSE uses several layers of security to minimize any risks of errors or vulnerabilities in the system. Additionally, we stress that CEPESC does not have access to the core implementation of TSE or any of their systems, and that `libharpia` is just a dry cryptographic API that could be used in any other system. Therefore, it is TSE's responsibility to use `libharpia` correctly.

3. Library Design

In order to facilitate the inclusion of new algorithms and interchangeability of the designated algorithms, the architecture of the library has to be modular. This is achieved by working with a layered design based on libsodium [Denis 2013] and NaCl [Bernstein et al. 2012], where in the lowest layer we have the implementations of all the primitives of the library, AVX2, SSE2, reference implementations, and any other applicable instruction extension set. A middle layer that standardizes cryptographic operations, by doing what is common to, for instance, any stream cipher independent of what implementation one is using. And a top layer that is responsible for interfacing with the user with a higher level of abstraction. Providing support to these three layers, there is a set of utilities, composed of constant time implementations of commonly used functions,

mainly buffer comparison and a centralized entropy source. This is the `core`, and it is highlighted in blue in Figure 1.

A fourth layer is above the `core`, which is responsible for the actual user interface TSE uses. This layer exists solely to fulfill API retrocompatibility and facilitate the integration of our library in TSE's systems. This layer also has some utilities related to legacy functionalities such as an ASN.1 and AES implementations used in a key encryption protocol. In the future, a new API will be negotiated with TSE, and this fourth layer will not be necessary, which will improve maintainability.

The modularity comes from the interfaces between these layers. When a new primitive is added, by complying with the interface, turning that primitive into the designated primitive is a matter of changing pointers. Despite not much more code being necessary to allow a change of primitives during runtime, `libharpia` defines its primitives when compiling, in order to comply with our strict versioning premise.

The interface between bottom and middle layer allows written code in middle layer to abstract which implementation is being used. For instance, a stream cipher needs to present a function pointer that takes a nonce, a key and a length, and outputs a keystream. The middle layer will regard the function pointer presented, and not which implementation provided it (AVX2 or reference, for example). In the middle layer we adjust the primitives, to conform with the interface between top and middle layer, which is to say that enough information about the primitive needs to be passed to the top layer so that sanitizing user inputs can be done on the top layer, abstracting the primitive.

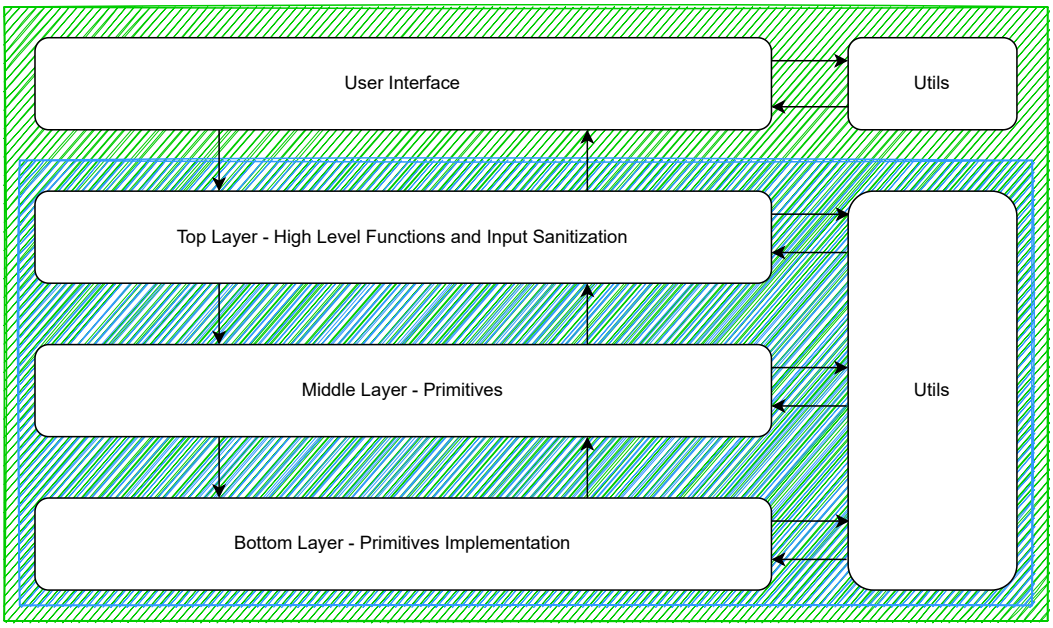


Figure 1. libharpia's architecture.

Based on this discussion, the main difference of this library when compared to the libraries we based our work on is the use of hybrid public-key protocols, including post-quantum algorithms. Another important difference is the nonce generation - `libharpia` generates a random nonce for each encryption while both libsodium and NaCl leaves the nonce management to the user. This choice protects the overall solution against common

implementation errors from the developers, such as nonce reuse.

Next, we describe a series of security principles adopted in the design and implementation of `libharpia`.

3.1. No Heap allocation and memory management

Abusing the heap is a very common vector of attack on secure systems. Improper handling of crashes, heap overflows, all contribute to nullifying security [Novark and Berger 2010], specially for modern solutions that involve co-hosting, or simply sharing resources with an adversary process. Nevertheless, this risk can be mitigated through careful implementation. In the development of a cryptographic library, it is more easily done by avoiding the use of heap allocation in the library, and relying on the stack.

Furthermore, to mitigate the impact of other types of side-channel memory attacks, we implement a stack memory management to ensure that every memory address that contained secret information (e.g., keys, plaintexts, cipher states) is erased as soon as it reaches its final use. This way, we seek to minimize the lifespan of sensible data in the machine.

3.2. No data flow from secrets to load addresses or branch conditions

Since 2005, it has been known that if any secret influence load addresses, a viable vector of attack is created by timing an unprivileged process in the same machine [Bernstein 2005]. Once these properties influence performance, different keys impact the adversarial process differently. Thus, by measuring this impact, cryptographic keys can be discovered almost instantly in implementations that use, for example, fast lookup tables [Pointcheval 2006, Tromer et al. 2010]. We avoid any implementations where secrets are suspected or known to affect performance.

Another source of timing side channel attacks is secrets being used as branch conditions. Admittedly, the choice of primitives helps here, since there are primitives that by design favor constant time. But more importantly, the implementations used are checked using `valgrind` [Langley 2010].

3.3. Centralizing randomness - cryptographic secure random number generator

Centralizing randomness gives us safety through maintainability. By focusing our attentions on a single source of randomness, we can ensure that any fix or improvement made on it is reflected across all the library. No primitive in our library gets entropy through any other means. By having changes in the source code follow a strict suite of tests on the quality of the random number generator, this culminates in a safer library.

3.4. Nothing is decrypted unless it first survives authentication

Encryption without authentication is not offered. No data is decrypted if authentication fails. Additionally, authentication verification occurs in constant time.

3.5. Strict versioning

As the library is designed to work only in the Brazilian Elections system, it will be used in a very closed environment. The library is designed to be able to communicate only

with others in the same version, so when a new version is released, all components of the system must be upgraded to this version.

This approach avoids downgrade attacks, where an attacker can exploit security flaws in older versions even if the system is running an up-to-date version [Alashwali and Rasmussen 2018]. Although no backwards compatibility is too rigid for decentralized environments, this strategy is reasonable to implement in our scope as the library distribution to the components is controlled by a centralized entity, the TSE.

3.6. Key generation

All persistent (non-ephemeral) assymetric key-pairs used in the library are generated using an external program to the library. This program communicates with a cryptographic token developed by CEPESC, the PCPv2 (Portable Cryptographic Platform). PCPv2 is an USB device with secure storage and processing capabilities and provides, among other cryptographic functionalities, a physical True Random Number Generator.

A sequence of random numbers is provided by PCPv2's TRNG and combined with the machine entropy pool as a seed to the key generation program's PRNG. As the random number generation for keys is a critical part of every cryptographic implementation and weak RNGs are the primary source of numerous practical attacks on secure systems, we opted to have a conservative approach using TRNGs. Furthermore, the double-source of entropy approach is able to mitigate risks of entropy pool poisoning.

4. Cryptographic Primitives

One of the main aspects of `libharpia` is that its security is founded on the security of the primitives and protocols used. A rule of thumb for choosing good cryptographic primitives was to avoid secret and untrusted ones. Additionally, we focused on maximizing performance in software and minimizing memory usage. As we aim to 256 bits security for the standard operations of the library, some primitives were also not considered.

We chose the combination Chacha20-Poly1305 for authenticated encryption with associated data, as it is referred in [Nir and Langley 2018]. The first question that comes in mind is "Why not use AES?". The simple answer is that Chacha performs better than AES on software, and in most CPUs without hardware acceleration, with greater security margin. Chacha20 is also intrinsically time safe. Another aspect of concern is the higher complexity of AES, which still is subject to hidden relations, for instance, recently some surprising properties were found for its key schedule [Leurent and Pernot 2021]. Poly1305 is a message authentication code and its security is based on the security of the underlying algorithm, which means that if one could break Poly1305 than one would also break Chacha20. We also note that CEPESC's researchers extensively studied the security of ChaCha [Coutinho and Neto 2021].

The standard choice for hash function in `libharpia` is Blake2 [Saarinen and Aumasson 2015], which is an improved version of Blake, a finalist of the SHA3 competition. Blake2 has seen widespread use across cryptographic libraries (e.g., OpenSSL, WolfSSL and Sodium), and it is also used as a RNG for the Linux Kernel. It is faster than MD5, SHA1, SHA2 and SHA3 while been as safe as SHA3.

`libharpia` really stands out amongst other libraries in its use of post-quantum cryptography through hybrid protocols. In the classical part, we use Curve448 for

key exchange and its Edwards counterpart Ed448 for digital signature [Hamburg 2015]. Curve448 uses Solina's prime $p = 2^{448} - 2^{224} - 1$ which has the special format $p = 2^{2\phi} - 2^{\phi} - 1$, $\phi = 224$, resembling the golden ratio, from where it received the *Goldilocks* nickname, and allows for fast Karatsuba multiplication. Moreover, the curve has fast and friendly implementations for 32 and 64 bits platforms. The curve is considered safe under all the parameter from `https://safecurves.cr.yp.to/` and it is adopted by the new TLS 1.3. Although its expected security is 224 bits, bellow our aim, it is high enough for our purposes, since the hybrid protocols ensure that both primitives have to be compromised in any successful attack, and our choices for post-quantum primitives are believed to have at least 256 security.

With the post-quantum cryptography standardization in its final stages, we can clearly see the prevalence of lattice based cryptography. As a matter of fact, only two out of seven submissions are not lattice based, and the one base on multivariate equations was recently completely broken [Beullens 2022]. Our choice was to use the CRYSTALS family, that is, Dilithium for digital signature and Kyber for key encapsulation. Both algorithms rely on the Learning With Errors problem, which were proposed in [Regev 2005].

The CRYSTALS constructions were built from module lattices, that is, lattices drawn from more structured spaces, in this case, subspace of modules. The digital signature algorithm Dilithium [Ducas et al. 2018] is based on the Module-LWE problem, where the error is taken uniformly over the correspondent ring. Its signature scheme is based on the Fiat-Shamir With Aborts proposed by Lyubashevsky [Lyubashevsky 2009] and has several types of encoding in order to reduce its key and signature sizes. It performs better and has smaller public key plus signature size compared to all lattices submissions. Kyber [Bos et al. 2018] is a key encapsulation algorithm also based on the Module-LWE problem. It achieves indistinguishability under adaptive chosen ciphertext attack using the Fujisaki-Okamoto transform over a public key encryption algorithm.

As a final note, it is important to mention that recently the stated security of all LWE finalist were reduced accordingly to the Matzov center [MATZOV 2022]. In essence, the reduction were due improvement in the algorithms from the known dual lattice attack on LWE. In our view, this does not compromise the whole scheme because a change of parameters (one of the main advantages of considering modules) could mitigate the reduction. That said, as the NIST competition also decided for the CRYSTALS family, there is no current reason to change our original choice.

5. API

In this section, we present some of the functions provided by `libharpia`. The functions that are not listed here are simple variations of the presented ones and exist mainly for backward compatibility. We refer to Section 6 to more details about the protocols implemented in these functions.

The library's API is a requirement defined by TSE. Because of that, some of the elements of the API are present only to fulfill retrocompatibility requirements. Let's look at encryption as an example. We start with a initialization function:

```
rv = init_encryption(k,NULL,0,ct,ctl,pk,NULL);
```

As the name implies, `init_encryption` is responsible for any preparation necessary

for encryption, more precisely, `init_encryption` executes a hybrid key exchange using the keys that were generated preemptively, as explained in Section 3.6. The parameter `k` is a pointer to a memory area that will receive the symmetric key from the hybrid key exchange. `ct`, `ctl` and `pk` are pointers to, respectively, the encapsulated symmetric key, its length and the public key. Both `NULL` parameters are innocuous and the `0` could be any 64 bit unsigned int, which are present only to achieve retrocompatibility with the old API calls. Any `rv` different than `0` indicates an error. These error are designed to not give any critical information about the failure, but rather indicate what element was at fault, for example, the ASN.1 encoding or incompatible library versions. The `rv` behaves the same for all the functions.

Next, the encryption function is quite straight forward:

```
rv = encrypt(k,p,pl,c,cl,NULL);
```

Giving as input the symmetric key `k`, that one have already obtained using `init_encryption`, the plaintext `p` and its length `pl`, one will get the ciphertext on pointer `c`. The ciphertext has the same size as the plaintext plus 44 bytes and its length has to be inputted in `cl`, as a sanity check. Those 44 bytes are used for authentication with associated data and are composed of 28 bytes of metadata such as the library version, plaintext length and nonce, and the remainder 16 bytes for the authentication tag. Algorithm 3, presented in Section 6.4, explains the Authenticated Encryption scheme And like all other relevant length values, it is provided as a macro in our headers. Again, we note that `NULL` is only used for retrocompatibility.

Signing has a much simpler interface. Taking a buffer and a private key, and outputting the signature:

```
rv = sign_buffer(b,bl,s,sl,pk);
```

where `b` is a pointer to a buffer to be signed, `bl` is its length, `s` is a pointer where the signature will be outputted, `sl` is the length of the signature, an input serving as a sanity check, and `pk` is a pointer to the private key.

Naturally, we have `init_decryption`, `decrypt` and `verify_buffer`, which are analogous to the functions presented so far. We also provide a key derivation function:

```
rv = derive_key(sk,dk,salt,saltl,info,infol,NULL);
```

where `sk` is a pointer to the symmetric key, `dk` is a pointer to the memory address where the derived key will be outputted, `salt` is a pointer to the salt used in the key derivation process, `saltl` is a pointer to its length which should be a minimum of 128 bits, `info` is a pointer to any additional information to the key derivation (e.g., voter ID), `infol` is the length of all the info added.

6. Cryptographic Protocols

In this section, we present four cryptographic protocols implemented in our library. Each of those protocols is directly linked to functions provided by the API.

The signature and the key exchange are public key cryptography protocols and the classical methods for those protocols (RSA, elliptic curves) are being threatened with the

possibility of quantum computing and Shor's algorithm [Roetteler et al. 2017]. Therefore, we decided to use hybrid public key protocols, combining classical elliptic curves with candidates algorithms approved for the third and final round of NIST post-quantum cryptography standardization process [Alagic et al. 2020]. The rationale behind the hybrid approach is the conservative one: the protocols design aim to have at least the security of the most secure of both primitives in the classical and the post-quantum setting. In this way, we are protected in the non-quantum scenario with the well-known security of the elliptic curves and possibly in the quantum scenario with algorithms that relies on problems believed to be hard-to-solve in quantum computers.

The third protocol is a symmetric-key derivation protocol that derives a new symmetric key from a previous one using some associated data, used in the generation of the encryption keys of individual data of the voters. And, finally, the fourth protocol is the symmetric-key authentication encryption/decryption scheme, that must be used with keys derived from the key-exchange protocol or from the symmetric-key derivation protocol.

6.1. Hybrid Signature

The hybrid signature protocol is a simple signature protocol that combines elliptic curve signature using Ed448 and post-quantum signature using the lattice algorithm Dilithium5 presented in Section 4. The hybrid signature $S_m^{hy} = Sign^{ec}(m, SK_s^{ec})||Sign^{pq}(m, SK_s^{pq})$ of a message m is a concatenation of the elliptic curve signature of m and the post-quantum signature of the same message using the sender's elliptic curve and post-quantum secret keys SK_s^{ec},SK_s^{pq}, respectively.

To verify the signature S_m^{hy} of a message m, the verifier splits the signature in S_m^{ec}, S_m^{pq} and verify each part separately with the sender's public keys PK_s^{ec}, PK_s^{pq} such that the signature is only valid if both signatures are valid for m. For a forgery in this protocol, the adversary needs to forge both signatures, so this protocol is at least as secure as the most secure of both primitives.

The verification is always done for both signatures even if the first one already fails. This strategy is used to achieve constant-time verification and therefore avoid leaking which part of the hybrid signature failed the verification.

6.2. Hybrid Key Exchange

The hybrid key-exchange protocol combines elliptic curves cryptography with quantum-resistant lattice based public-key cryptographic algorithms. The protocol is designed such that an attack to the protocol able to recover the derived key must attack both the classical and the post-quantum primitive, so the security of the protocol is dependent of the major security of both primitives. In other words, if the attacker has a quantum computer we are protected by the post-quantum algorithms. Conversely, if the chosen post-quantum algorithms turn out not to be secure, we still have classical (and well established) security of the elliptic curves. The protocol presented below is based on the draft of Internet Engineering Task Force about hybrid post-quantum key encapsulation methods (PQ KEM) for transport layer security 1.2 (TLS) [Campagna and Crockett 2021].

For the classical part of the protocol, the library implements an ephemeral elliptic curve Diffie-Hellman using the curve described in Section 4 with the X448 implementation. In this protocol, we obtain the receiver elliptic curve public key PK_r^{ec} and generate

a ephemeral key pair $(SK^{ec}_{eph}, PK^{ec}_{eph})$ using a cryptographic secure random number generator. Then, we compute the elliptic curve shared secret $ss^{ec} = X448(SK^{ec}_{eph}, PK^{ec}_{r})$.

In the post-quantum part, we use the Key Encryption Mechanism provided in Kyber1024, presented in Section 4. Given the receiver post-quantum public key PK^{pq}_r we compute the ciphertext C^{pq} and the post-quantum shared secret ss^{pq} with $C^{pq}, ss^{pq} = ENC(PK^{pq}_r)$ where ENC is the Kyber1024.CCAKEM Key Encapsulation function.

With both shared-secrets, we compute the shared 256-bit symmetric key K_E using the Blake2 function, presented in Section 4, as the keyed hash function $H_K(.)$ such that $K_E = H_{ss^{pq}}(H_{ss^{ec}}(PK^{ec}_{eph}||C^{pq}||PK^{ec}_r||PK^{pq}_r))$. The protocol returns the symmetric encryption key K_E, the ephemeral elliptic curve public key PK^{ec}_{eph} and the post-quantum encapsulation ciphertext C^{pq}. After encryption of a plaintext with the key K_E, it is necessary to send to the receiver the PK^{ec}_{eph} and C^{pq}, so it can recover K_E for decryption. The protocol is described in Algorithm 1.

The receiver recovers the symmetric key K_E in three steps. First, one executes the elliptic curve Diffie-Hellman using his secret key SK^{ec}_r and the received ephemeral public key PK^{ec}_{eph} obtaining the first shared secret $ss^{ec} = X448(SK^{ec}_r, PK^{ec}_{eph})$. Using the ciphertext C^{pq} and his private key SK^{pq}_r, one is able to recover the second shared secret $ss^{pq} = DEC(C^{pq}, SK^{pq}_r)$ using the Kyber1024.CCAKEM Key Decapsulation function. With both shared secrets it can be obtained $K_E = H_{ss^{pq}}(H_{ss^{ec}}(PK^{ec}_{eph}||C^{pq}||PK^{ec}_r||PK^{pq}_r))$.

Algorithm 1 Hybrid Key Exchange

function KEY EXCHANGE(PK^{ec}_r, PK^{pq}_r) ▷ Inputs both public keys of the receiver
  $ss^{ec}, PK^{ec}_{eph} \leftarrow ECDHE(PK^{ec}_r)$ ▷ Ephemeral Diffie-Hellman
  $ss^{pq}, C^{pq} \leftarrow ENC(PK^{pq}_r)$ ▷ Post-quantum key encapsulation
  $K_E \leftarrow H_{ss^{pq}}(H_{ss^{ec}}(PK^{ec}_{eph}||C^{pq}||PK^{ec}_r||PK^{pq}_r))$ ▷ Derived symmetric key
  sends PK^{ec}_{eph}, C^{pq} to receiver
  return K_E ▷ Symmetric Key for encryption
end function

6.3. Symmetric-Key Derivation

This protocol is a Key Derivation Function (KDF) that uses a symmetric-key K_S obtained by the Hybrid Key Exchange protocol to generate one or more secure symmetric keys with no need to call the more expensive asymmetric key protocol. This protocol is based on the *Extract-then-Expand KDF* construction, the same used in the design of HKDF and that has a sound security proof [Krawczyk 2010]. The proposed construction has two differences when compared to HKDF: (i) it has a fixed sized output length, not requiring an extra parameter and a loop in its implementation; and (ii) it uses the hash function Blake2 directly instead of HMAC. Note that both choices improve performance. In particular, HMAC requires a hash function being called two times, therefore, this KDF is at least two times faster. Here, performance was the main goal, as the protocol is designed for a specific use case that consists of the encryption of the biometric data of voters. More specifically, a new key is derived for each voter, yielding in a very computationally intensive process (we have more than 150 million voters in Brazil).

For each voter v, we derive a new symmetric key K_v, using a randomly generated 16-byte salt S_v and the voter's unique registration number R_v. We used the keyed hashing version $H_K(.)$ of Blake2, our choice of hash function, as presented in Section 4. The details are shown in Algorithm 2.

Algorithm 2 Symmetric Key Derivation

function KEY DERIVATION(K_S, R_v, S_v)
  $prk \leftarrow H_{S_v}(K_S)$ ▷ The hash function is used as a randomness extractor
  $K_v \leftarrow H_{prk}(R_v)$ ▷ The hash function is used as a PRF
  return K_v
end function

6.4. Authenticated Encryption

The library's authenticated encryption scheme uses the stream cipher Chacha20 and authentication algorithm Poly1305 described in Section 4. We based our proposed protocol on the ideas contained in the RFC-8439 [Nir and Langley 2018] by the Internet Research Task Force, adapting it for our algorithms.

Our authenticated encryption algorithm receives as input a N-byte long plaintext P_N and a 256-bit symmetric key K_E. As a first step, a 12-byte $Nonce$ is randomly generated using a cryptographic secure random number generator. This random nonce generation aims to prevents nonce misuse, a common problem for implementations using stream ciphers. The plaintext P_N is encrypted with the encryption key K_E and the generated nonce resulting in the ciphertext C_N.

For authentication purposes, we derive a new 256-bit authentication key K_A using the library's keyed hash function with random generated nonce as input and the key K_E. The authenticated array D is the result of the concatenation of the library's version, the nonce used in encryption, the ciphertext C_N and the ciphertext length N as a 8-byte little-endian unsigned integer. We compute the 128-bit authentication tag T applying the MAC algorithm Poly1305 with key K_A at the array D and returns $D||T$. A pseudocode for the authenticated encryption protocol is presented at Algorithm 3.

The random generation of the nonce implies a probability of 2^{-48} for collision. Nevertheless, in the library's use case, a new symmetric key is obtained at each encryption - from the key-exchange protocol (6.2) or from the key derivation scheme (6.3). In this scenario, the key/nonce tuple will have a much smaller probability of being reused.

In the authenticated decryption scheme, the function receives the array D containing the ciphertext and the associated information (library version, nonce and ciphertext length) as presented above, the authentication tag T and the symmetric encryption key K_E. The first step is to recover the authentication key K_A using the nonce and K_E. We check if T is a valid tag for D using K_A, and if not, the algorithm stops and returns an error. Otherwise, we continue the procedure checking if the length N included in D corresponds to the received ciphertext length. Only if both checks pass, we proceed to the decryption of C_N using K_E with our stream cipher and returns the result P_N.

This protocol aims to guarantee that invalid key/tag/ciphertext sets will not be decrypted, reducing the computational cost of invalid function calls.

Algorithm 3 Authenticated Encryption

function AUTHENTICATED_ENCRYPTION(P_N, K_E)
$Nonce \leftarrow CSRNG(12)$ ▷ 12-byte nonce randomly generated
$C_N \leftarrow ENC_{K_E}(P_N, Nonce)$ ▷ Encrypts with stream cipher
$K_A \leftarrow H_{K_E}(Nonce)$ ▷ Auth Key from Nonce and Encryption Key
$N_{le8} \leftarrow num\_to\_8\_le\_byte(length(P_N))$ ▷ Length of plaintext
$V \leftarrow Version()$ ▷ Version of the Library
$D \leftarrow V||Nonce||C_N||N_{le8}$ ▷ Concatenate Associated Data and ciphertext
$T \leftarrow Auth_{K_A}(D)$ ▷ Compute authentication tag
return $D||T$
end function

7. Conclusion

In this work, we presented `libharpia`, a new library to be applied in Brazilian elections. Throughout the work, we provided all design choices and cryptographic primitives, showing that `libharpia` is aligned with the best practices of secure implementations and cryptography. In addition, we detailed the main cryptographic protocols, justifying their designs based on well established literature and standards. We also provided a detailed description of how the library is used in practice, and its API. Through the use of post-quantum cryptography and hybrid protocols, we showed that `libharpia` provides a security advantage when compared with other cryptographic libraries available. Despite all technical characteristics of `libharpia`, the most important aspect of this work is that it consists of another step towards transparency and auditability of the Brazilian elections, describing in a clear and open fashion all details of the library. As a final note, we remember that `libharpia` is present in TSE's TPS, therefore, the source code of the library is available for any group that desires to analyze it and test its security in practice. Additionally, we remark that there is an intention of open-sourcing this library in the near future.

Acknowledgements We would like to thank all TSE's professionals that were involved in the specification and testing of the library.

References

Alagic, G., Alperin-Sheriff, J., Apon, D., Cooper, D., Dang, Q., Kelsey, J., Liu, Y.-K., Miller, C., Moody, D., Peralta, R., et al. (2020). Status report on the second round of the nist post-quantum cryptography standardization process. *US Department of Commerce, NIST*.

Alashwali, E. S. and Rasmussen, K. (2018). What's in a downgrade? a taxonomy of downgrade attacks in the tls protocol and application protocols using tls. In *International Conference on Security and Privacy in Communication Systems*, pages 468–487. Springer.

Alessandre, S. (2021). Add support for x509 certs with nist p384/256/192 keys. `https://patchwork.kernel.org/project/linux-crypto/cover/20210316210740.1592994-1-stefanb@linux.ibm.com/`.

Aranha, D. F., Barbosa, P., Cardoso, T. N. C., Araújo, C. L., and Matias, P. (2019). The return of software vulnerabilities in the brazilian voting machine. *Comput. Secur.*, 86:335–349.

Bernstein, D. J. (2005). Cache-timing attacks on aes. `https://cr.yp.to/antiforgery/cachetiming-20050414.pdf`.

Bernstein, D. J., Lange, T., and Schwabe, P. (2012). The security impact of a new cryptographic library. In *Progress in Cryptology – LATINCRYPT 2012*, Lecture Notes in Computer Science, pages 159—-176. Springer.

Beullens, W. (2022). Breaking rainbow takes a weekend on a laptop. *IACR Cryptol. ePrint Arch.*, page 214.

Bos, J. W., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J. M., Schwabe, P., Seiler, G., and Stehlé, D. (2018). CRYSTALS - kyber: A CCA-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P*, pages 353–367. IEEE.

Brunazo Filho, A., Carvalho, M., Teixeira, M., Simplicio Jr, M., and Fernandes, C. (2015). Auditoria especial no sistema eleitoral 2014. *Simpósio Brasileiro em Segurança da Informação e de Sistemas Computactionais, XV, Florianópolis. Anais... Florianópolis: SBC*, pages 511–522.

Campagna, M. and Crockett, E. (2021). Hybrid post-quantum key encapsulation methods (pq kem) for transport layer security 1.2 (tls). Internet-draft, IETF Secretariat. `https://www.ietf.org/archive/id/draft-campagna-tls-bike-sike-hybrid-07.txt`.

Coimbra, R. C. M., Monteiro, J. R. M., and da Silva Costa, G. (2017). Registro impresso do voto, autenticado e com garantia de anonimato. *Anais do XVII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 666–681.

Coutinho, M. and Neto, T. C. S. (2021). Improved linear approximations to ARX ciphers and attacks against chacha. In *Advances in Cryptology - EUROCRYPT 2021*, volume 12696 of *Lecture Notes in Computer Science*, pages 711–740. Springer.

Denis, F. (2013). The sodium cryptography library. `https://download.libsodium.org/doc/`.

Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., and Stehlé, D. (2018). Crystals-dilithium: A lattice-based digital signature scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):238–268.

Gallo, R., Kawakami, H., Dahab, R., Azevedo, R., Lima, S., and Araujo, G. (2010). T-DRE: a hardware trusted computing base for direct recording electronic vote machines. In *Twenty-Sixth Annual Computer Security Applications Conference, ACSAC 2010, Austin, Texas, USA, 6-10 December 2010*, pages 191–198. ACM.

Hamburg, M. (2015). Ed448-goldilocks, a new elliptic curve. *IACR Cryptol. ePrint Arch.*, page 625.

Krawczyk, H. (2010). Cryptographic extraction and key derivation: The HKDF scheme. In *Advances in Cryptology - CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 631–648. Springer.

Langley, A. (2010). ctgrind—checking that functions are constant time with valgrind. https://github.com/agl/ctgrind.

Leurent, G. and Pernot, C. (2021). New representations of the AES key schedule. In *Advances in Cryptology - EUROCRYPT 2021*, volume 12696 of *LNCS*, pages 54–84. Springer.

Lyubashevsky, V. (2009). Fiat-shamir with aborts: Applications to lattice and factoring-based signatures. In *Advances in Cryptology - ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 598–616. Springer.

MATZOV (2022). Report on the Security of LWE: Improved Dual Lattice Attack.

Monteiro, J., Lima, S., Rodrigues, R., Alvarez, P., Meneses, M., Mendonça, F., and Coimbra, R. (2019). Protegendo o sistema operacional e chaves criptográficas numa urna eletrônica do tipo t-dre. In *Anais do IV Workshop de Tecnologia Eleitoral*, pages 1–12. SBC.

Nir, Y. and Langley, A. (2018). ChaCha20 and Poly1305 for IETF Protocols. RFC 8439.

NIST (2022). Nist announces first four quantum-resistant cryptographic algorithms. https://www.nist.gov/news-events/news/2022/07/nist-announces-first-four-quantum-resistant-cryptographic-algorithms.

Novark, G. and Berger, E. D. (2010). Dieharder: securing the heap. In *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010*, pages 573–584. ACM.

Pointcheval, D. (2006). *Topics in Cryptology – CT-RSA 2006: The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2005, Proceedings*. LNCS sublibrary: Security and cryptology. Springer.

Regev, O. (2005). On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 84–93. ACM.

Roetteler, M., Naehrig, M., Svore, K. M., and Lauter, K. (2017). Quantum resource estimates for computing elliptic curve discrete logarithms. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 241–270. Springer.

Saarinen, M. O. and Aumasson, J. (2015). The BLAKE2 cryptographic hash and message authentication code (MAC). *RFC*, 7693:1–30.

Tromer, E., Osvik, D. A., and Shamir, A. (2010). Efficient cache attacks on aes, and countermeasures. In *Journal of Cryptology*, volume 23, pages 37–71.

TSE (2021). Novas urnas eletrônicas contarão com certificação da icp-brasil. https://www.tse.jus.br/comunicacao/noticias/2021/Julho/novas-urnas-eletronicas-contarao-com-certificacao-da-icp-brasil.

van de Graaf, J. and Custódio, R. (2002). Tecnologia eleitoral e a urna eletrônica–relatório sbc 2002. *Disponıvel em http://www.sbc. org.br/index.php*.

Artigos publicados em 2023

Automated security proof of SQUARE, LED and CLEFIA using the MILP technique

Gabriel Cardoso de Carvalho[1], **Tertuliano Souza Neto**[2], **Thiago do Rêgo Sousa**[2]

[1]Instituto de Computação - Universidade Federal Fluminense (UFF) - Niterói - RJ

[2]CEPESC, Agência Brasileira de Inteligência, Brasília - DF

Abstract. *Provable security in cryptography is extremely relevant nowadays, since it is regarded as the basis for the proposal of new ciphers. In that sense, the designers of new ciphers have to find ways to prove that the proposed cipher is secure against the most pertinent forms of attack. Being safe against differential and linear cryptanalysis is still considered the bare minimum standard for any new cipher. In the last decade, a great deal of attention has been given to automated ways of proving the security of ciphers against both forms of attacks, the original one being generating mixed linear integer programs that model the given cipher in such a way that, by solving it, we are able to know the minimum number of rounds necessary for the cipher to be secure. In this paper, we apply this technique in the well known block ciphers LED, SQUARE and CLEFIA, and compare the results with the original security claims.*

1. Introduction

Symmetric cryptography is a very active research field since it is one of the basic building blocks for secure communication. In particular, many new block ciphers have been developed, the most famous probably being data encryption standard (DES) and advanced encryption standard (AES) algorithms. Their design usually takes into account recent developments of new attacks, and the designers try to build ciphers which are resistant to particular classes of attack. Resistance against differential and linear cryptanalysis was and still is one of the main goals of new cipher designs, but proving such things might be a bit cumbersome for the designer.

In the last decade a couple of automated ways of proving under which conditions a cipher is resistant to these attacks was developed. Along with the Boolean Satisfiability Problem [Mironov and Zhang 2006], mixed linear integer programming (MILP) is one very important tool as an alternative approach to analyze the security bounds of a cipher [Mouha et al. 2011]. In particular, the MILP method is used to prove security against both differential and linear cryptanalysis by means of solving a linear mixed integer program which is directly connected to the working mechanism of the cipher. Since its introduction, MILP has been used to prove the security of many ciphers, including SIMON, PRESENT, LBlock [Sun et al. 2014], LIZARD [Karthika and Singh 2023], and Midori64 [Zhao et al. 2020].

Besides its use as a means of providing provable security, MILP based methods have also been used in a variety of applications in modern cryptanalysis, such as searching for integral distinguishers [Xiang et al. 2016] and looking for differential and linear trails [Fu et al. 2016]. Moreover, much work has been done in improving the speed of the MILP model [Zhou et al. 2019] as well as using it in cryptography design [Pal et al. 2023].

Therefore, since its introduction in 2011, the use of MILP models in the area of cryptology has been relevant, which in turn justifies our choice of topic. However, applying the MILP modeling from [Mouha et al. 2011] requires the underlying cipher to have some properties. First of all, the cipher needs to be word-based (nibble, byte) and its internal mechanisms being based on S-boxes, XOR operations, linear permutation and/or three-forked branches. This is the case of AES and Enocoro, which were analyzed in the original paper of [Mouha et al. 2011].

SQUARE [Daemen et al. 1997] is one classic cipher that is still relevant nowadays for being the precursor of the AES, as well as having introduced the SQUARE attack - a type of cryptanalysis directed to AES-like ciphers, such as KIASU-BC [Dobraunig et al. 2016] and Midori64 [Wardhana and Indarjani 2019]. The LED cipher [Guo et al. 2011] is well known for being one of the first lightweight ciphers (Light Encryption Device), which are commonly used in Internet of Things and other embedded systems, and for being target of extensive cryptanalysis since its creation although remaining secure. CLEFIA [Katagi and Moriai 2011] is also a lightweight cipher. In fact, it is one of the standardized lightweight encryption algorithms of ISO/IEC 29192-2:2019.

Due to their relevancy, these three block ciphers will be the subject of this paper. We will apply the MILP modeling to find the minimum number of active S-boxes that should be activated during a differential/linear attack and compare with their original security claims. MILP has also been applied before to light encryption device (LED) and CLEFIA, but in another context. [Hadipour et al. 2022] applied MILP to get faster distinguishes for LED (8 rounds) and CLEFIA (10 and 11 rounds) using the division property and [Derbez and Lambin 2022] used MILP to attack 11 rounds of CLEFIA in the key-recovery setting.

The rest of the paper is organized as follows. In Section 2, we describe the MILP model from [Mouha et al. 2011] and how it is applied in the context of linear and differential cryptanalysis. A brief description of SQUARE, LED and CLEFIA algorithms focusing on the important mechanism required by MILP will be given in Section 3. Then, in Section 4, we give the details on the application of the method in each algorithm, reporting the results and making appropriate comparisons. Section 5 concludes the paper and give further directions of research.

2. The MILP model

Since its introduction by [Mouha et al. 2011], MILP modeling of ciphers has been extensively applied to prove security against linear and differential cryptanalysis in many different encryption algorithms. One of the advantages of this method is its generality and its adaptability. In particular, it can be applied to any word-based cipher constructed using linear permutation layers, S-boxes, XOR operations and/or three-forked branches, as is the case of Enocoro [Watanabe and Kaneko 2007] and AES.

Another important advantage of this technique is that the workload of the cryptanalist is reduced to simply describing the cipher by means of linear equations expressing how the input and output chunks of words relate to each other. Once this step is done, the equations, and restrictions generated by the internal mechanisms of the ciphers are put into a linear solver, and an integer linear equation representing the number of active Sboxes is minimized. This gives us an almost automatic way of proving security

bounds against both linear and differential cryptanalysis.

It is also worth noting that there are other techniques suitable for finding the minimum number of active Sboxes in a cipher but MILP requires less programming efforts from the cryptanalist (see [Mouha et al. 2011, page 3]).

We will give a brief idea of how the method works in the context of differential cryptanalysis and refer to the reader to look at [Mouha et al. 2011] for more information. To make the explanation clear we will describe the case where the input and output of the operations in a cipher are represented in the level of bytes.

When working with differential cryptanalysis we are interested in the difference of byte strings and a key concept for applying MILP is the difference vector. For a string $\Delta = (\Delta_0, \Delta_1, \ldots, \Delta_{n-1})$ of n bytes, the difference vector $x = (x_0, x_1, \ldots, x_{n-1})$ is such that each $x_i = 0$ if the byte $\Delta_i = 0$ and $x_i = 1$ otherwise. This is because the only important information here is whether the byte difference is zero or not, regardless of its value.

All input and output variables of the ciphers are treated as unique variables, independent of the round we are in (in case the cipher is based on rounds). For each operation (linear transformation, XOR, etc) we have a set of equations describing it. Every operation involving input and output variables is analyzed, and possibly generates a restriction involving these variables, which can be written in terms of inequalities. In the end, an objective function involving only the input variables that enter the Sboxes is created. Gathering these variables together with these inequalities and the objective function, we can generate what is commonly known as a MILP problem, which we can type into one of many available solvers and find out the answer. That answer tells us what is the minimum number of active Sboxes that comprises with the working mechanisms of the cipher being analyzed.

For a given operation $\mathcal{O}$, let $(x^{\mathcal{O}}_{in_1}, x^{\mathcal{O}}_{in_2}, \ldots, x^{\mathcal{O}}_{in_M})$ be the input differences and $(x^{\mathcal{O}}_{out_1}, x^{\mathcal{O}}_{out_2}, \ldots, x^{\mathcal{O}}_{out_N})$ be the corresponding output differences. Another useful concept is of the *differential branch number* $\mathcal{B}_D$. It is defined as the minimum of the sum of the number of active bytes in the input and output of the operation, excluding the trivial case where all input and output bytes are zero. It describes how the input and output bytes are related in a certain operation. Using $\mathcal{B}_D$ and the input and output bytes of $\mathcal{O}$, the inequalities describing it in the MILP model are the following:

$$
\begin{aligned}
x^{\mathcal{O}}_{in_1} + x^{\mathcal{O}}_{in_2} + \ldots x^{\mathcal{O}}_{in_M} + x^{\mathcal{O}}_{out_1} + x^{\mathcal{O}}_{out_2} + \cdots + x^{\mathcal{O}}_{out_N} &\geq \mathcal{B}_D d^{\mathcal{O}}, \\
d^{\mathcal{O}} &\geq x^{\mathcal{O}}_{in_1}, \\
d^{\mathcal{O}} &\geq x^{\mathcal{O}}_{in_2}, \\
&\ldots\ldots \\
d^{\mathcal{O}} &\geq x^{\mathcal{O}}_{in_M}, \\
d^{\mathcal{O}} &\geq x^{\mathcal{O}}_{out_1}, \\
d^{\mathcal{O}} &\geq x^{\mathcal{O}}_{out_2}, \\
&\ldots\ldots \\
d^{\mathcal{O}} &\geq x^{\mathcal{O}}_{out_N},
\end{aligned}
\tag{1}
$$

where $d^{\mathcal{O}}$ is a *dummy* binary variable, which is zero when all input and output is zero and 1 otherwise. This is necessary to avoid the trivial case where all input and output are zero and to avoid the resolution of several integer linear programs as was needed in [Bogdanov 2011].

If $\mathcal{O}$ is the XOR operation, then $\mathcal{B}_D = 2$ and we can write the first inequality of (1) as

$$x^{\oplus}_{in_1} + x^{\oplus}_{in_1} + x^{\oplus}_{out_1} \geq 2d^{\oplus}, \tag{2}$$

where $d^{\oplus}$ is the corresponding dummy variable. The equations describing a linear transformation in a cipher are similar, except that the differential branching number changes depending on the operation.

For the case of linear cryptanalysis we only need to change the differential branch number (DBN) to the linear branch number (LBN) $\mathcal{B}_L$ and also treat the three-forked branch as an operation whose input and output variables need to be taken into account.

With this in mind, we describe in the next section a collection of algorithms that we found suitable for the application of the MILP technique due to their structure not only in terms of what operations they involve, but also in terms of how the algorithm is applied.

3. Chosen block ciphers suitable for MILP applications

All of the following ciphers were chosen due to their academic relevancy and for having two properties required for MILP to work.

The first required property is being word-based, which means that all operations work only in a word to word level: there are no bit-wise shifts/rotations or bit-wise permutations. For instance, if a cipher contains an XOR operation of two 8-bit words in a given part and a 3-bit shift operation in another, than it is not word-based. Using words of different size throughout the cipher is also unwanted.

The second property we need is having Sboxes as the only non-linearity source. This means that the cipher's design adds non-linearity through a function or lookup table that substitutes a given input by an output in such a way that is non-linear. This is needed because the MILP model intends to count the minimum number of such Sboxes that must be activated in order to attack the cipher, either in a differential or a linear context. Other sources of non-linearity cannot be dealt through this method nor can they be ignored by it.

Therefore, we looked for the most relevant ciphers with these two properties that were not already tested (which excluded the most prominent example, AES) and ended up with two similar ciphers in design. The first one is the SQUARE cipher, well known for being the predecessor of the Rijndael (AES), by the same authors [Daemen et al. 1997], and for introducing the Square Attack, a new form of cryptanalysis at the time. The second algorithm targeted is LED, one of the first in a class of block ciphers labeled as lightweight which means that this cipher had software implementation speed as the main focus, while still maintaining enough security to be used. While based on a different design, the CLEFIA cipher also has the desired property of being word-based and using S-boxes as the only non-linearity source. It is the security standard for the products of the

Sony company[1] and has been subjected to many different analysis by the cryptography community since its introduction.

The following subsections describe the main mechanisms of each cipher and, since the key schedule part does not play a role in the application of MILP, we leave it out of the algorithm descriptions below.

3.1. SQUARE

The block cipher square was introduced in [Daemen et al. 1997] and it has block and key length of 128 bits. Its design takes into account resistance against differential and linear cryptanalysis. In terms of performance, careful choice of the building blocks was made to allow for efficient implementations on many processors.

Each round of SQUARE comprises four different transformations. A linear transformation applied separately to each state row, an Sbox (nonlinear transformation), a byte transformation (basically interchanging of columns and rows of the state) and a bit-wise round key addition. All of them operating on a 4 by 4 array of bytes.

The whole cipher SQUARE is defined as eight rounds which are applied after a key addition.

3.2. LED

LED is a 64-bit block cipher dedicated to compact hardware implementation. It was presented in [Guo et al. 2011] and is based on the design principles of AES which allows one to obtain simple bounds in terms of the number of active S-boxes during encryption. Its design features the well known AES operations such as S-boxes, ShiftRows and MixColumns. To understand the working mechanisms of LED, one can think of the cipher state as arranged in a 4 by 4 grid where each nibble represents an element from $GF(2^4)$. Field multiplication is done with the polynomial X^4+X+1. The initial state of the cipher is a 4 by 4 grid whose entries are formed by the 16 four-bit nibbles of the message. The cipher has no key-schedule and nibbles of subkeys are added to the state using bit-wise exclusive-or. The key size can be either 64 or 128 bits and the number of steps during encryption varies (8 applications of the step function for 64 bit and 12 applications for 128 bit key).

The step function is the core for encryption and it is defined as the application of the AddConstants (bitwise shift applied just to some constants, which does not affect the state), SubCells (applies S-Boxes to each state nible), ShiftRows (rotation of rows to the left) and MixColumnsSerial (multiplication of array state vector by a matrix) operations in order. The SubCells operation is the most important one since it is the one that introduces non-linearity to the cipher. For more details see [Guo et al. 2011, Section 2.1].

3.3. CLEFIA

CLEFIA was presented in [Katagi and Moriai 2011]. It is a highly efficient block cipher and achieves a good performance both in hardware and software. It has 128 bit block size with key lengths of 128, 192 and 256. It is constructed based on a generalized Feistel structure using two 32 bit F functions at each round. These functions are

[1] `https://www.sony.net/Products/cryptography/clefia/`. Accessed 06/06/2023.

different compared to traditional Feistel structure and they require more rounds. On the other hand, the F-functions are smaller and plural F-functions can be processed simultaneously, surpassing the disadvantage of having more rounds. Two F functions are used in each round, but they have the same design, except for the choice of the internal S-boxes. The S-boxes are used to introduce non-linearity through the Diffusion Switching Mechanism, which led to strong resistance against certain attacks studied in the original paper [Katagi and Moriai 2011].

In addition, CLEFIA allows for flexible implementation in both software and hardware.

4. Results

In this section, we study the security of SQUARE, LED and CLEFIA against differential and linear cryptanalysis through the semi-automated MILP technique. The reasoning for choosing these algorithms is that they share two properties that are needed for applying the original MILP method as was explained in the beginning of Section 3.

For each of these ciphers, the main objective was to obtain the MILP program in such a way that a solver is able to find out what is the minimum number of Sboxes needed to be activated for a differential or linear attack to be applied. Both SQUARE and LED have the same MILP program for both linear and differential cryptanalysis because the differential and linear branch numbers are equal for each operation of these ciphers, as they are all SPNs. On the other hand, CLEFIA is a Feistel cipher, which implies the presence of three-forked branch operations. This operation is only relevant for linear cryptanalysis. We have a similar situation for the XOR operation, which is needed for differential cryptanalysis but not for linear cryptanalysis.

In the following subsections, we describe and compare our results to the existing literature. All of our tests were conducted by generating the MILP program of each relevant number of rounds for each cipher. The resulting programs were then fed to the open-source MILP solver SCIP [Bestuzheva et al. 2021] which returned the minimum amount of Sboxes necessary for each of the aforementioned cases. For replication purposes, our MILP programs as well as the results for the SCIP execution can be found at `https://github.com/MfMhj3uNy5gfp4Z/MILP-results/tree/master`.

We ran all tests on a AMD FX(tm)-8320 Eight-Core Processor 3.50 GHz and, in general, the results were obtained in less than 20 seconds for up to 10 rounds for all ciphers, ramping up to anywhere between 5 to 10 minutes for cipher LED and CLEFIA for 18 rounds. Since LED has 48 rounds, it lakes long periods of time to finish all of the above 19 rounds and thus, for any attempt to execute past a established time constraint given by the available computational capacity we forcefully stopped the solver if the expected results based on its design were already attained (see the observation in Table 2).

4.1. SQUARE

The SQUARE cipher was constructed with the famous wide trail design strategy [Daemen 1995], in which the choices for the construction of the cipher are based in two criteria:

1. The maximum difference propagation probability δ (security against differential

attacks), as well as linear propagation probability λ (security against linear attacks) of the chosen Sboxes must be as low as possible.
2. The linear parts must not leave any trail with few active Sboxes.

This design strategy has been used across the board in the creation of block ciphers and specially Substitution Permutation Networks. The most prominent ones being the SQUARE and Rijndael [Daemen and Rijmen 2002] ciphers. The latter became known as AES and is the American national standard for symmetric encryption, as well as one of the most used block ciphers in the world.

In this context, SQUARE presents an Sbox with $\delta = 2^{-6}$ and $\lambda = 2^{-3}$, which means that any attack that activates at least 22 Sboxes is unfeasible since it would imply $\frac{1}{\delta^{22}} = 2^{132}$ plaintext-ciphertext pairs for a differential attack whereas there are only 2^{128} possible pairs for the SQUARE cipher.

As for a linear attack, at least 43 Sboxes need to be activated to make the attack unfeasible, since $\frac{1}{\lambda^{43}} = 2^{129}$ is bigger than the available 2^{128} pairs.

Table 1 shows our results on the cipher SQUARE. It is possible to notice that 4 rounds are enough to guarantee security against differential cryptanalysis and 6 rounds suffices for resistance against linear cryptanalysis.

Table 1. Minimum amount of Sboxes to be active in any given differential or linear attack for the SQUARE cipher, obtained through the MILP program.

# rounds	1	2	3	4	5	6	7	8	9	10	11	12
# Sboxes	1	5	9	25	26	30	34	50	51	55	59	75

4.2. LED

The LED cipher is also termed as an AES-like cipher since it uses the same kind of operations as the AES. AES-like ciphers all use as basis the wide trail strategy. Therefore, we should expect to find the same minimum Sboxes per round as the SQUARE. Furthermore, the states and the branch numbers of the operations are remarkably similar.

The main difference though is that LED uses a 64 bit state divided into words of size 4, also called nibbles. Consequently, the Sbox has to be different and, accordingly, has different values for the differential probability δ and linear probability λ. The Sbox is reused from the PRESENT cipher [Bogdanov et al. 2007] and it has $\delta = 2^{-2}$ and $\lambda = 2^{-7}$.

Since the block state has 64 bits, the maximum amount of plaintext-ciphertex pairs to be used for an attack has to be at most 2^{64}. Table 2 shows that the threshold or number of rounds for guaranteed security through the MILP method against differential attacks is such that the corresponding minimum number of Sboxes s is given by $\frac{1}{\delta^s} = 2^{2s} \geq 2^{64}$. The minimum number of rounds is 7 ($2^{2\times 34} = 2^{68} \geq 2^{64}$). For the linear case, one has $\frac{1}{\lambda^s} = 2^{7s} \geq 2^{64}$ and hence 4 rounds are enough ($2^{7\times 25} = 2^{175} \geq 2^{64}$).

Although we are handling the cipher as composed simply by rounds, it is actually composed by steps, which in turn are composed of 4 consecutive rounds. The 64-bit key version of LED has 6 steps while the 128-bit key version has 12 steps. Thus, using the aforementioned calculations, LED needs two steps to be secure against differential attacks and only one step to be secure against linear attacks.

Table 2. Minimum amount of Sboxes to be active in any given differential or linear attack for the LED cipher, obtained through the MILP program. These results were obtained by the solver before finishing its entire search. Although that means it would be theoretically possible finding a better result, the wide trails design strategy used to construct LED supports that the best estimate could not be smaller than the ones obtained bellow.

# rounds	# Sboxes	# rounds	# Sboxes	# rounds	# Sboxes	# rounds	# Sboxes
1	1	13	76	25	151*	37	226*
2	5	14	80	26	155*	38	230*
3	9	15	84	27	159*	39	234*
4	25	16	100	28	175*	40	250*
5	26	17	101	29	176*	41	251*
6	30	18	105	30	180*	42	255*
7	34	19	109	31	184*	43	259*
8	50	20	125*	32	200*	44	275*
9	51	21	126*	33	201*	45	276*
10	55	22	130*	34	205*	46	280*
11	59	23	134*	35	209*	47	284*
12	75	24	150*	36	225*	48	300*

4.3. CLEFIA

The CLEFIA cipher uses a different base structure in comparison to the other two ciphers we have seen so far. While SQUARE and LED are Substitution Permutation Networks, CLEFIA is a generalized Feistel cipher. Generalized Feistel ciphers characteristically have their state divided into a power of two number d and only half of those are used as input to a non-linear function F whose output is XORed to the other half of the state. Then, a simple rotation is applied to all parts in the state.

This internal structure implies that there are three-forked branches before the input is sent to F. This operation is relevant to the linear attack but not to the differential one, which in turn contrasts with the XOR operation, that is relevant only to the differential attack.

Notably though, Table 3 shows that both cases are remarkably similar, the only difference being the minimum amount of Sboxes necessary for 6 rounds, in which differential attacks need 12 Sboxes, while linear attacks require 11. This result was also obtained by the authors [Katagi and Moriai 2011] through ad-hoc computational search.

CLEFIA also uses two Sboxes (S_0 and S_1) while both SQUARE and LED use just one. For the Sbox S_0 we have $\delta_0 = 2^{-4.67}$ and $\lambda_0 = 2^{-4.38}$ and for S_1 we have $\delta_1 = \lambda_1 = 2^{-6}$. Since both are used in parallel, the analysis can become complex and even gets outside the scope of the MILP method.

Therefore, we chose $\delta = \delta_0$ and $\lambda = \lambda_0$ for all calculations, since they are the best ones. This implies that we assume all Sboxes that are active are S_0. Although unrealistic in a practical sense, this still conforms to the main purpose of the method, which is to find a lower bound to the number of rounds necessary to guarantee security.

We now use the same calculations as for SQUARE and LED to compute the

Table 3. Minimum amount of Sboxes to be active in any given differential or linear attack for the CLEFIA cipher, obtained through the MILP program.

Differential				Linear			
# rounds	# Sboxes	# rounds	# Sboxes	# rounds	# Sboxes	# rounds	# Sboxes
1	0	10	18	1	0	10	18
2	1	11	20	2	1	11	20
3	2	12	24	3	2	12	24
4	6	13	24	4	6	13	24
5	8	14	25	5	8	14	25
6	12	15	26	6	11	15	26
7	12	16	30	7	12	16	30
8	13	17	32	8	13	17	32
9	14	18	36	9	14	18	36

minimum amount of rounds necessary. A CLEFIA block has 128 bits, thus any attack that requires over 2^{128} plaintext-ciphertext pairs is unfeasible. Then, the amount of rounds is such that the associated minimum number of active Sboxes s conforms to $\frac{1}{\delta^s} = 2^{4.67 \times s} \geq 2^{128}$ for differential and $\frac{1}{\lambda^s} = 2^{4.38 \times s} \geq 2^{128}$ for linear attacks.

For the differential case, 16 rounds are enough, since it requires 30 Sboxes to carry an attack which satisfies $2^{4.67 \times 30} = 2^{140.1} \geq 2^{128}$. Although the linear probability is lower than the differential, the same amount of rounds is necessary for linear attacks as the same 30 Sboxes satisfy $2^{4.38 \times 30} = 2^{131.4} \geq 2^{128}$.

Besides the use of two Sboxes, CLEFIA also has two different linear operations, which makes it harder to explore cancellations and thus increases the minimum amount of Sboxes necessary to apply an attack. Unfortunately, the word-based MILP method lacks flexibility to deal with these intricacies.

5. Conclusion

In this paper, we applied the MILP method to show proof of security against differential and linear cryptanalysis for the ciphers SQUARE, LED and CLEFIA. All three algorithms are secure as expected, SQUARE needing only 6 rounds to be safe against differential and 4 to be safe against linear, while LED needs 2 steps (7 rounds) for differential and 1 step (4 rounds) for linear. CLEFIA requires 6 rounds to be secure from differential attacks and 11 for linear.

The results obtained here are consistent with the ones presented by the original authors, i.e., the same minimum amounts of Sboxes expected are equal to the ones obtained by our MILP model for all three ciphers. Indeed, the authors of both SQUARE and LED got their results in a theoretical manner. Through the wide trail design strategy they showed that the minimum amount of Sboxes grows in a fixed rate following a $(1, 4, 4, 16)$ pattern (ex. LED with 4,5,6,7,8 rounds have 25, 26, 30, 34, 50 minimum Sboxes, which agrees with the results of both Tables 2 and 1). On the other hand, the authors of CLEFIA use a computer program to conduct an ad-hoc search to find the minimum amounts of Sboxes, since it's design does not have a theoretical result backing it up. Therefore, the MILP model is a formidable tool for proving the security against linear and differential

cryptanalysis for certain types of ciphers.

Although the word-based MILP method is useful for some ciphers, its use is limited since it is unable to give a more accurate lower bound to the number of Sboxes for more complex designs, such as the use of more than one linear operation in parallel in the CLEFIA cipher.

For that reason, further research involves exploring more complete ways to develop MILP programs, such as the use of the bitwise MILP method which is capable of dealing with the intricacy of CLEFIA, as well as other ciphers that contain bitwise linear operations, such as DES, ARIA, Twofish, FEAL and Serpent.

References

Bestuzheva, K., Besançon, M., Chen, W.-K., Chmiela, A., Donkiewicz, T., van Doornmalen, J., Eifler, L., Gaul, O., Gamrath, G., Gleixner, A., Gottwald, L., Graczyk, C., Halbig, K., Hoen, A., Hojny, C., van der Hulst, R., Koch, T., Lübbecke, M., Maher, S. J., Matter, F., Mühmer, E., Müller, B., Pfetsch, M. E., Rehfeldt, D., Schlein, S., Schlösser, F., Serrano, F., Shinano, Y., Sofranac, B., Turner, M., Vigerske, S., Wegscheider, F., Wellner, P., Weninger, D., and Witzig, J. (2021). The SCIP Optimization Suite 8.0. Technical report, Optimization Online.

Bogdanov, A. (2011). On unbalanced feistel networks with contracting mds diffusion. *Des. Codes Cryptography*, (59(1-3)):35–58.

Bogdanov, A., Knudsen, L. R., Leander, G., Paar, C., Poschmann, A., Robshaw, M. J. B., Seurin, Y., and Vikkelsoe, C. (2007). Present: An ultra-lightweight block cipher. In Paillier, P. and Verbauwhede, I., editors, *Cryptographic Hardware and Embedded Systems - CHES 2007*, pages 450–466, Berlin, Heidelberg. Springer Berlin Heidelberg.

Daemen, J. (1995). *Cipher and hash function design strategies based on linear and differential cryptanalysis*. PhD thesis, Doctoral Dissertation, March 1995, KU Leuven.

Daemen, J., Knudsen, L., and Rijmen, V. (1997). The block cipher square. In *Fast Software Encryption: 4th International Workshop, FSE'97 Haifa, Israel, January 20–22 1997 Proceedings 4*, pages 149–165. Springer.

Daemen, J. and Rijmen, V. (2002). *The design of Rijndael*, volume 2. Springer.

Derbez, P. and Lambin, B. (2022). Fast milp models for division property. *IACR Transactions on Symmetric Cryptology*, pages 289–321.

Dobraunig, C., Eichlseder, M., and Mendel, F. (2016). Square attack on 7-round kiasu-bc. In *Applied Cryptography and Network Security: 14th International Conference, ACNS 2016, Guildford, UK, June 19-22, 2016. Proceedings 14*, pages 500–517. Springer.

Fu, K., Wang, M., Guo, Y., Sun, S., and Hu, L. (2016). Milp-based automatic search algorithms for differential and linear trails for speck. In *Fast Software Encryption: 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers 23*, pages 268–288. Springer.

Guo, J., Peyrin, T., Poschmann, A., and Robshaw, M. (2011). The led block cipher. In *Cryptographic Hardware and Embedded Systems–CHES 2011: 13th International Workshop, Nara, Japan, September 28–October 1, 2011. Proceedings 13*, pages 326–341. Springer.

Hadipour, H., Nageler, M., and Eichlseder, M. (2022). Throwing boomerangs into feistel structures: Application to clefia, warp, lblock, lblock-s and twine. *Cryptology ePrint Archive*.

Karthika, S. and Singh, K. (2023). Cryptanalysis of stream cipher lizard using division property and milp based cube attack. *Discrete Applied Mathematics*, 325:63–78.

Katagi, M. and Moriai, S. (2011). The 128-bit blockcipher clefia. Technical report.

Mironov, I. and Zhang, L. (2006). Applications of sat solvers to cryptanalysis of hash functions. In *Theory and Applications of Satisfiability Testing-SAT 2006: 9th International Conference, Seattle, WA, USA, August 12-15, 2006. Proceedings 9*, pages 102–115. Springer.

Mouha, N., Wang, Q., Gu, D., and Preneel, B. (2011). Differential and linear cryptanalysis using mixed-integer linear programming. In *International Conference on Information Security and Cryptology*, pages 57–76. Springer.

Pal, D., Chandratreya, V. P., and Chowdhury, D. R. (2023). Efficient algorithms for modeling sboxes using milp. *arXiv preprint arXiv:2306.02642*.

Sun, S., Hu, L., Wang, P., Qiao, K., Ma, X., and Song, L. (2014). Automatic security evaluation and (related-key) differential characteristic search: application to simon, present, lblock, des (l) and other bit-oriented block ciphers. In *Advances in Cryptology–ASIACRYPT 2014: 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, ROC, December 7-11, 2014. Proceedings, Part I 20*, pages 158–178. Springer.

Wardhana, D. and Indarjani, S. (2019). Square attack on 4 round midori64. In *AIP Conference Proceedings*, volume 2168. AIP Publishing.

Watanabe, D. and Kaneko, T. (2007). A construction of light weight panama-like keystream generator. *IEICE Technical Report*.

Xiang, Z., Zhang, W., Bao, Z., and Lin, D. (2016). Applying milp method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In *Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I 22*, pages 648–678. Springer.

Zhao, H., Han, G., Wang, L., and Wang, W. (2020). Milp-based differential cryptanalysis on round-reduced midori64. *IEEE Access*, 8:95888–95896.

Zhou, C., Zhang, W., Ding, T., and Xiang, Z. (2019). Improving the milp-based security evaluation algorithm against differential/linear cryptanalysis using a divide-and-conquer approach. *Cryptology ePrint Archive*.

Artigos publicados em 2024

Cutting dimensions in the LLL attack for the ETRU post-quantum cryptosystem

Augusto Miguel Camillo Silva[1], **Thiago do Rêgo Sousa**[2], **Tertuliano Souza Neto**[2]

[1]Universidade Federal de Juiz de Fora (UFJF) - Juiz de Fora - MG

[2]CEPESC, Agência Brasileira de Inteligência, Brasília - DF

Abstract. *NTRU is one of the most important post-quantum cryptosystems nowadays, based on polynomial rings with coefficients in $\mathbb{Z}$. Among its variants, the ETRU cryptosystem utilizes Eisenstein integers $\mathbb{Z}[\omega]$, where ω is a primitive cube root of unity. We explore this cryptosystem and introduce a new lattice based on May's technique, which proposes reducing the original lattice dimension to enable attacks with increased complexity. This new lattice allowed us to recover the private key of the ETRU system for a dimension that was not yet possible using current lattice reduction techniques over the original lattice.*

1. Introduction

Public key cryptography, initially proposed by Diffie and Hellman [Diffie W 1976], led to the development of various cryptographic systems, which currently protect a significant portion of digital communication. With the advancement of quantum computing, these classical public key cryptography algorithms like RSA [Rivest RL. 1978], elliptic curve cryptography (ECC) [Neal 1987], and Digital Signature Algorithm (DSA) [NIST 2019], are at risk of being broken in polynomial time by quantum computers using algorithms like Shor's [Shor 1994]. Post-quantum cryptography aims to ensure cryptographic security in the era of quantum computing.

The Post-Quantum Cryptography Standardization [NIST] is an initiative led by the National Institute of Standards and Technology in the United States with the goal of developing cryptographic standards that are secure against attacks from quantum computers. As of the latest update, the process was in its third round, with a reduced list of candidates considered for final standardization, including algorithms like Kyber, NTRU, and Classic McEliece.

The NTRU system was introduced by Hoffstein, Pipher, and Silverman, is an efficient public key cryptosystem based on polynomial rings with integer coefficients [Hoffstein et al. 1998]. NTRU is notable for its arithmetic operations of quadratic complexity, being significantly faster than RSA and ECC. It is based on the difficulty of solving certain lattice problems, such as finding the shortest vector in a convolutional lattice [May A 2001], making it resistant to quantum attacks. However, NTRU may suffer from decryption failures, although proper parameter selection can mitigate this issue.

Several variants of NTRU have been proposed to improve its security and efficiency, such as GNTRU which uses Gaussian integers, CTRU which is based on binary fields, QTRU using quaternion algebra, ETRU based on Eisenstein integers, among others [Sonika Singh 2016].

The ETRU system was introduced in [Monica Nevins 2010] as an extension of the original NTRU. A subsequent work by [Katherine Jarvis 2015] highlighted its superior speed, smaller key sizes, and simplicity in binary messaging compared to NTRU. Khaterine's study also compared the efficiency and security of ETRU and NTRU against meet-in-the-middle attacks and lattice attacks. In [Karbasi and Atani 2015], a new system called ILTRU was introduced as an extension of ETRU, exploiting properties of structured lattices to achieve high efficiency and security based on ideal lattices, with the established hardness of R-SIS[Lyubashevsky and Micciancio 2009] and R-LWE[Regev 2009] problems.

Subsequently, [S. Lyu and Ling 2020] deepened the understanding of the characteristics of algebraic lattices, emphasizing the appropriate design and performance limits of reduction algorithms. [Zhu and Tian 2021] compare the performance and security of NTRU and ETRU signature algorithms and argue that ETRU is faster.

On the security side of the ETRU system, [Katherine Jarvis 2015] proposed a lattice base attack on the private key that requires using base reduction techniques over a basis of dimension $4n$, where $n-1$ is the degree of the ETRU polynomials used to construct the system. We extend the original attack by introducing a new lattice that has vectors of smaller dimensions as was done in [May 1999] for the original NTRU system. We observe through a simulation study that, even after reducing the original lattice dimension, it is still possible to find the system's private key. The proposed attack can, in fact, recover the private key for $n = 61$ on a personal computer, something that was currently not feasible using the original lattice from [Katherine Jarvis 2015]. This suggests that ETRU has a lower level of security than expected by using its original lattice, highlighting the need for further analysis when choosing its parameter to ensure adequate security.

The rest of the paper is organized as follows. In Section 2 we introduce the NTRU system and in Section 3 its ETRU variant. The proposed key recovery attack using a lattice with a smaller dimension is developed in Section 4 and Section 5 concludes giving directions for further research.

2. The NTRU Cryptosystem

The NTRU public key cryptosystem depends on three integer parameters (n, p, q) such that $n \geq 1$, $\gcd(n, q) = \gcd(p, q) = 1$ and q is much larger than p. The primary arithmetic operations in the NTRU cryptosystem involve computations over polynomials defined in the rings $\mathcal{R}$, $\mathcal{R}_p$, and $\mathcal{R}_q$ as follows:

$$\mathcal{R} = \frac{\mathbb{Z}[x]}{(x^n - 1)}, \mathcal{R}_p = \frac{(\mathbb{Z}/p\mathbb{Z})[x]}{(x^n - 1)}, \mathcal{R}_q = \frac{(\mathbb{Z}/q\mathbb{Z})[x]}{(x^n - 1)}.$$

It can be observed that the ring $\mathcal{R}$ is associated with the other two rings. Specifically, for any polynomial $a(x)$ in $\mathcal{R}$, it can be associated with an element in $\mathcal{R}_p$ or $\mathcal{R}_q$ by reducing its coefficients modulo p or q, respectively.

A polynomial $a(x) \in \mathcal{R}$ is termed a ternary polynomial if its coefficients belong to the set $\{-1, 0, 1\}$. In addition, $a(x)$ can be associated with an element in $\mathcal{R}_p$ or $\mathcal{R}_q$ by reducing its coefficients modulo p or q, respectively

Given d_1 and d_2 positive integers. We define $\mathcal{T}(d_1, d_2)$ as the subset of ternary polynomial in $\mathcal{R}$ as follows:

$$\mathcal{T}(d_1, d_2) = \left\{ \begin{array}{l} a(x) \in \mathcal{R}, \\ a(x) \text{ has } d_1 \text{ coefficients equal to } 1, \\ a(x) \text{ has } d_2 \text{ coefficients equal to } -1, \\ \text{remaining coefficients of } a(x) \text{ are } 0. \end{array} \right\}$$

In the NTRU cryptosystem, operating with parameters (n, p, q, d), key generation, encryption and decryption are defined as follows:

1. **Key Generation:** Generate two ternary polynomials at random, $f(x) \in \mathcal{T}(d+1, d)$ and $g(x) \in \mathcal{T}(d, d)$ such that there exist two polynomials $f_p(x) \in \mathcal{R}_p$ and $f_q(x) \in \mathcal{R}_q$ satisfying $f(x)f_p(x) = 1 \in \mathcal{R}_q$ and $f(x)f_q(x) = 1 \in \mathcal{R}_q$. Then compute the polynomial:

   $$h(x) = f_q(x) * g(x) \in R_q,$$

   where $*$ denotes polynomial multiplication in R_q, i.e., a cyclic convolution product as defined in [Hoffstein et al. 1998, Section 1.1]. The polynomial $h(x)$ is the public key and the pair $(f(x), f_p(x))$ is the private key.

2. **Encryption:** Let $m(x) \in \mathcal{R}_p$ be a plaintext and choose, at random, a ternary polynomial $r(x) \in \mathcal{T}(d, d)$. The encrypted message is:

   $$e(x) \equiv ph(x) * r(x) + m(x) \mod q.$$

   Notice that the ciphertext $e(x)$ belongs to the ring $\mathcal{R}_q$.

3. **Decryption:** To decrypt the ciphertext first compute:

   $$a(x) \equiv f(x) * e(x) \mod q.$$

   Then the reduction modulo p gives the desired plaintext

   $$b(x) \equiv f_p(x) * a(x) \mod p.$$

Due to the randomness of the polynomial $r(x)$, NTRU operates as a probabilistic cryptosystem. This means that a message $m(x)$ can be encrypted into multiple ciphertexts $ph(x) \cdot r(x) + m(x)$, each depending on the particular instance of $r(x)$. However, this introduces a potential vulnerability in the NTRU cryptosystem, as certain ciphertexts may fail to decrypt correctly back to the original message, a scenario referred to as decryption failure. Attacks documented in the literature exploit such decryption failures [Howgrave-Graham et al. 2003, Gama and Nguyen 2007, Jaulmes and Joux 2000], underscoring the necessity for careful parameter selection.

3. The ETRU Cryptosystem

ETRU is a lattice-based cryptosystem which is a variant of NTRU, constructed using truncated polynomials with coefficients in the ring of Eisenstein integers $\mathbb{Z}[\omega]$. The ring of Eisenstein integers is the set of complex numbers of the form $a + b\omega$ with $a, b \in \mathbb{Z}$, where ω is a primitive cube root of unity.

3.1. Describing ETRU

Let q be a nonzero element of $\mathbb{Z}[\omega]$. The set $(q) = \{rq \mid r \in \mathbb{Z}[\omega]\}$ forms the ideal in $\mathbb{Z}[\omega]$ generated by q. Denote by $\mathbb{Z}_q[\omega]$ the set of residue classes of the quotient ring $\mathbb{Z}[\omega]/\langle q \rangle$. For instance, $\mathbb{Z}_2[\omega]$ represents a field with four elements, namely, $\{0, 1, \omega, \omega + 1\}$. For any $z = a + b\omega \in \mathbb{Z}[\omega]$ we can define its norm by

$$|z| = a^2 - ab + b^2.$$

To reduce the probability of decryption failure, we shall represent the set of residues centered around 0, i.e., for an integer n, we have

$$\mathbb{Z}_n = \begin{cases} \left\{-\dfrac{n-1}{2}, \cdots, \dfrac{n-1}{2}\right\} & \text{if } n \text{ is odd,} \\ \left\{-\dfrac{n}{2} + 1, \cdots, \dfrac{n}{2}\right\} & \text{if } n \text{ is even.} \end{cases}$$

To set the ETRU parameters, we initially select two relatively prime elements in $\mathbb{Z}[\omega]$, p and q such that $|q|$ is much larger than $|p|$. This is necessary to make the polynomial inversion algorithms modulo p and q more efficient. It is preferable to choose both elements as primes or powers of primes. We choose a positive integer n (preferably prime) and set:

$$\mathcal{R} = \frac{\mathbb{Z}[\omega][x]}{(x^n - 1)}; \qquad \mathcal{R}_p = \frac{\mathbb{Z}_p[\omega][x]}{(x^n - 1)}; \qquad \mathcal{R}_q = \frac{\mathbb{Z}_q[\omega][x]}{(x^n - 1)} \tag{1}$$

Note that an element $f \in \mathcal{R}$ is a polynomial $f_0 + f_1 x + ... + f_{n-1} x^{n-1}$ where each coefficient f_i is an Eisenstein integer $f_i = a_i + b_i\omega$. Similarly, a polynomial $f \in \mathcal{R}_p$ (or $\mathcal{R}_q$) if and only if each coefficient $f_i \in \mathbb{Z}_p[\omega]$ (or $\mathbb{Z}_q[\omega]$). We define a rotation of $f \in \mathcal{R}$ as the polynomial

$$x^k f(x) = f_0 x^k + f_1 x^{k+1} + ... + f_{n-1} x^{n+k-1} \in \mathcal{R},$$

for an integer $k \in \mathbb{Z}$.

We've chosen $p = 2$ throughout the process, which offers numerous advantages in encoding binary messages into elements of $\mathcal{R}_p$, as mentioned in [Katherine Jarvis 2015].

Fixing $0 < r < 1$, we define the sets $\mathcal{L}_f$, $\mathcal{L}_q$, and $\mathcal{L}_\varphi$ of polynomials as the subsets of $\mathcal{R}$ containing approximately nr non-zero coefficients selected from the set $\mu_6 = \{\pm 1, \pm\omega, \pm\omega^2\}$ of units of $\mathbb{Z}[\omega]$ as follows.

Let k be the nearest integer to nr. The polynomials in $\mathcal{L}_f$ consist of all polynomials with k nonzero entries.

The polynomials in $\mathcal{L}_q$ and $\mathcal{L}_\varphi$ should be divisible by $x - 1$ modulo q. To achieve this, we select s, the nearest multiple of three to rn, and randomly pick s tuples of coefficients, each being $\pm\{1, \omega, \omega^2\}$. These tuples are then distributed across the coefficients while preserving the order of the chosen tuples.

For fixed n, p, q,r key generation, encryption and decryption in the ETRU system work as follows:

1. **Key Generation:** To generate the keys, choose two random polynomials $f(x) \in \mathcal{L}_f$ and $g(x) \in \mathcal{L}_g$. The polynomial $f(x)$ must have inverses mod p and mod q. Let $f_p(x) \in \mathcal{R}_p$ and $f_q(x) \in \mathcal{R}_q$ be the inverses of $f(x)$ under mod p and mod q, respectively. Thus, $f_p(x) \cdot f(x) \equiv 1 \mod p$ and $f_q(x) \cdot f(x) \equiv 1 \mod q$. Then, calculate $h(x) = f_q(x) \cdot g(x) \mod q$.
   The pair of polynomials $(f(x), f_p(x))$ is the private key and the polynomial $h(x)$ is the public key in the ETRU cryptosystem.

2. **Encryption:** To encrypt a message $m(x) \in \mathcal{R}_p$ we first choose a random polynomial $\varphi(x) \in \mathcal{L}_\varphi$ and then compute:
$$e(x) = p\varphi(x) \cdot h(x) + m(x) \mod q$$
   The polynomial $e(x) \in \mathcal{R}_q$ is the ciphertext.

3. **Decryption:** To decrypt the ciphertext $e(x)$ compute:
$$a(x) = f(x) \cdot e(x) \mod q$$
$$m(x) = f_p(x) \cdot a(x) \mod p$$
   We claim that the polynomial m is the original message that was encrypted above. In fact,
$$\begin{aligned} a(x) &= f(x) \cdot e(x) \mod q \\ &= f(x) \cdot (p\varphi(x) \cdot h(x) + m(x)) \mod q \\ &= pf(x) \cdot \varphi(x) \cdot h(x) + f(x) \cdot m(x) \mod q \end{aligned}$$
   The last equation holds if the coefficients of $pf \cdot \varphi \cdot h + f \cdot m$ are sufficiently small such that their values remain unchanged when reduced mod q. Now we compute
$$\begin{aligned} & f_p(x) \cdot a(x) \mod p \\ &= f_p(x) \cdot (pf(x) \cdot \varphi(x) \cdot h(x) + f(x) \cdot m(x)) \mod p \\ &= f_p(x) \cdot pf(x) \cdot \varphi(x) \cdot h(x) + f_p(x) \cdot f(x) \cdot m(x) \mod p \\ &= 0 + f_p(x) \cdot f(x) \cdot m(x) \mod p \\ &= m(x) \mod p \end{aligned}$$

In the next Section, we show how a key recovery attack is constructed for the ETRU system using lattices. After that, we introduced a new lattice that has a smaller dimension and can still be used for attacking the ETRU private key.

4. Key recovery attack

The coefficients of the public key $h(x)$ satisfy $f(x) * h(x) \equiv g(x) \mod q$. Thus, we can attack ETRU by solving this equivalence similarly to how we attack NTRU using lattices ([Hoffstein et al. 1998, Section 3.4]).

In the case of ETRU, a key $h(x)$ with parameters (n, q, p, r) generates a $4n-$dimensional lattice as we will show bellow. The vector corresponding to the pair of private keys (f, g) is a short vector in this lattice (in terms of the Euclidean norm), and therefore, we could discover the private key by finding a sufficiently short vector in the lattice. To achieve this, we use lattice basis reduction techniques, such as the LLL (Lenstra-Lenstra-Lovász) algorithm [Lenstra et al. 1982] or the BKZ (Block Korkin-Zolotarev) algorithm [Chen and Nguyen 2011].

4.1. ETRU lattice

Consider the isomorphism of additive groups $\varphi : \mathbb{Z}[\omega] \to \mathbb{Z}^2$ given by

$$\varphi(\alpha) = \varphi(a + b\omega) = (a, b).$$

For each element $\alpha \in \mathbb{Z}[\omega]$, we define the matrix $\langle\alpha\rangle$ that performs right multiplication in $\mathbb{Z}^2$

$$\langle\alpha\rangle = \begin{bmatrix} a & b \\ -b & a-b \end{bmatrix}$$

Let M be an $n \times n$ matrix with entries in $\mathbb{Z}[\omega]$. We define $\langle M\rangle$ as the $2n \times 2n$ matrix over $\mathbb{Z}$ by replacing each entry a_{ij} of M with $\langle a_{ij}\rangle$. Similarly, for any polynomial $f \in \mathcal{R}$, we define $\langle f\rangle$ as the application of the same operation over each coefficient of f.

For a given public key h for the ETRU system with parameters (n, q, p, r), let H represent the matrix formed by the coefficients of h and its $n-1$ rotations. Therefore,

$$\langle H\rangle = \begin{pmatrix} \langle h\rangle \\ \langle xh\rangle \\ \vdots \\ \langle x^{n-1}h\rangle \end{pmatrix}.$$

Then, the lattice of ETRU is defined as follows

$$L_{ETRU} = \begin{bmatrix} I_{2n} & \langle H\rangle \\ 0 & \langle qI_{2n}\rangle \end{bmatrix}, \tag{2}$$

where I_{2n} is the $2n$-dimensional identity matrix.

In ETRU, the private keys are associated with short vectors within L_{ETRU}. Indeed, the target vector (f, g) containing the private key associated with h, can be written as a linear combination of the rows of the matrix L_{ETRU} from (2)([Monica Nevins 2010, Section 8.1]). In other words, (f, g) belongs to the lattice generated by L_{ETRU}.

Using Gaussian heuristic [Nguyen 2010], the shortest vector expected from a lattice L of dimension N has length:

$$l = \sqrt{\frac{N}{2\pi e}} v^{1/N} \quad \text{where } v = det(L).$$

Thus, the shortest expected vector of the lattice L_{ETRU} of dimension $4n$ has length:

$$l_{ETRU} = \sqrt{\frac{4n}{2\pi e}} v^{1/4n} = \sqrt{\frac{4n}{2\pi e}} |q|^{2n/4n} = \sqrt{\frac{2n|q|}{\pi e}}.$$

The keys f and g each have rn non-zero entries in $\mu_6 = \{\pm 1, \pm\omega, \pm\omega^2\}$. In the lattice, each coefficient is viewed as $\{(\pm 1, 0), (0, \pm 1), (\pm 1, \pm 1)\}$. Thus, the norm of the target vector (f, g) lies between $\sqrt{2rn}$ and $\sqrt{4rn}$.

In cases where the norm of a vector (f, g) is maximized and the $|q|$ is sufficiently large, we obtain $\sqrt{4rn}$, which remains smaller than the expected shortest vector in the lattice. Hence, the likelihood of the pair (f, g) being found in the reduced basis lattice is high.

4.2. New ETRU lattice attack

The use of the BKZ (Block Korkin-Zolotarev) algorithm for lattice basis reduction becomes ineffective as the lattice dimension increases. This is because the execution time and complexity of BKZ grow exponentially with the dimension. To deal with this, we can apply May's idea [May 1999], which proposes reducing the lattice dimension. Specifically, May's idea involves cutting some coordinates of the vector g to reduce the problem's dimension for the NTRU system of [Hoffstein et al. 1998]. In what follows, we adapt this idea to attack the ETRU system in dimensions larger than those possible so far with the original lattice from (2).

Given the public key h and the corresponding lattice L_{ETRU}, we can define a new lattice L'_{ETRU} by excluding $k < 2n$ columns from the submatrix $\langle H \rangle$. Let $\langle H \rangle_k$ be the columns of $\langle H \rangle$ that are kept. The new lattice L'_{ETRU} can be expressed as:

$$L'_{ETRU} = \begin{bmatrix} I_{2n} & \langle H \rangle_k \\ 0 & \langle qI_{2n-k} \rangle \end{bmatrix} \tag{3}$$

Thus, the shortest expected vector of the lattice L'_{ETRU} of dimension $4n - k$ has a length of:

$$l'_{ETRU} = \sqrt{\frac{4n-k}{2\pi e}} v^{1/(4n-k)} \quad \text{where } v = |q|^{\frac{2n-k}{4n-k}}$$

Let g_k be the polynomial g with k coordinates removed. If $k < rn$, the norm of the target vector (f, g_k) lies between $\sqrt{2rn}$ and $\sqrt{4rn}$. If $k > rn$, the shortest vector has a norm smaller than $\sqrt{4rn}$.

[May 1999]'s attack works since the truncated short vector (lets say by removing the last coefficients of the g polynomial) is still a linear combination of the lattice vectors (truncated at the same positions that we removed from g). In fact, one could simply choose k coefficients of the polynomial g and remove it to create a new lattice and solve the SVP problem. This approach speeds up the search for the shortest vectors since the computational time for running BKZ is roughly proportional to the dimension of the lattice.

Removing columns to reduce the dimension of the lattice L'_{ETRU} may introduce some inaccuracies in the lattice structure. However, for specific values of the parameters n and q, this approximation is still effective for identifying short vectors that correspond to the private keys (f, g_k). Therefore, despite the introduced inaccuracies, the reduced lattice remains sufficiently informative to enable the identification of the private keys. The dimension of the reduced lattice L'_{ETRU}, although this approximation introduces some inaccuracies, for certain values of n and q it is sufficient to find a short vector that corresponds to the private keys (f, g_k).

The removal of columns can be done randomly, but the effectiveness of this attack can vary significantly depending on which columns are removed. The goal of May's attack is to reduce the dimension of the lattice without losing the vectors that contain the necessary information for the attack. If the wrong columns are chosen, the resulting reduced lattice may not exhibit the properties needed for a successful attack. In this work, the exclusion of the columns was done on the right side of the matrix $\langle H \rangle$.

Algorithm to find the private key:

Input: Integers n, q, a public key h for the ETRU system and a cut parameter $k > 0$.

Output: A set of potential private keys f^* corresponding to h.

1. Use h, q, n and k to construct the corresponding lattice L'_{ETRU} from (3). This is a matrix where the upper right corner is formed by the coefficients of the polynomial h where every line is just a circular shift by one of the previous line.
2. Apply the BKZ algorithm to reduce the lattice basis and get a matrix L'_{reduced} of dimension $4n \times (4n - k)$.
3. Use the first $2n$ coordinates of every line of L'_{reduced} to construct a list of vectors.
3. For each vector in the list of vectors from step 3 and each value $r \in \{0, 1, 2, \ldots, 2n - 1\}$, create a list of potential keys by rotating the vector r positions.

4.3. Experimental results

In [Monica Nevins 2010], the BKZ algorithm was applied to find a vector with the same norm as the target vector (f, g) corresponding to the private key of the ETRU system, in order to assess its security against lattice attacks. The results reported in their paper showed the viability of the attack for a fixed $q = 383$ and $n \leq 57$. In the highest degree achieve, i.e., $n = 57$, the success rate of the attack is about 20% percent. The authors reported that after $n = 57$, the BKZ attack using the original ETRU lattice from (2) consistently fails and we also observed this phenomenon in our simulation studies when trying to run for $n > 57$.

To go beyond $n = 57$, our strategy is to use the new lattice developed in Section 4.2. With a $k > 0$, the lattice dimension drops from $4n$ to $4n - k$ and we can hope to successfully execute BKZ in order to find the private key pair.

The results are reported in Table 1 where for each value of $n \in \{41, 47, 57, 61\}$ and the BKZ block used to run the attack. We report the value k used for cutting the dimension of the lattice and the success rate of the attack in 100 experiments, usually for $k \in \{1, \ldots, 2n - 1\}$ The time to run the attack is closely related to the lattice dimension $4n - k$ and the BKZ block. For small values of n we can use a block of 10 in the BKZ algorithm and it works for finding the private key. On the other hand, for $n = 57$ and 61 we needed to increase the block to 20, since running BKZ with a block of 10 did not succeeded in finding the private key.

For $n = 41$, running the attack with a cut of 59 already returned a success rate of 1%, in which case the dimension of the lattice decrease from 164 to 105, which is a great improvement in the complexity of the attack and shows the usefulness of [May 1999]'s idea. When we decrease the cut to 50 the success rate drastically increases to 69% and it achieves 100% for a cut of 21.

For the case where $n = 61$ we experimented with $k \in \{30, 31, 2n - 1\}$, since running BKZ in the lattice L'_{ETRU} with $k < 30$ is very likely to fail. For $k = 41$ and in case BKZ runs without error, the average time taken per experiment is less than 2 minutes. For each experiment we generated a new key pair and applied the algorithm of Section 4.2, recording whether or not we found the correct key in the list of potential keys f^*. Out of 100 independent experiments, we found the correct key in 3 of them,

n	41	47	57	61
BKZ block	10	10	20	20
cut k (sucess)	54 (6%)	54 (3%)	59 (1%)	49 (1%)
	50 (69%)	50 (53%)	50 (51%)	45 (7%)
	43 (94%)	43 (88%)	45 (94%)	*
	21 (100%)	27 (96%)	*	*

Table 1. Matrix NTRU private key attack for varying n and fixed $q = 383$. For some cut values we reported the sucess rate of the attack over 100 experiments.

indicating the usefulness of the L'_{ETRU} lattice in finding the private key. To run the attack successfully we needed to use a BKZ block of 20, which slows the basis reduction process but has the advantage of returning a shorter basis for the lattice. The success rate of the algorithm decreases as we increase the value of n, and this is due to the fact that running BKZ would need more computational resources. On the other hand, the attack presented here shows that the dimension reduction can be used to improve BKZ in this setup. This means that we do not need to work with the complete ETRU lattice to find the private key and this should be taken into account when analyzing its real security against lattice attack in a similar way that was done for the NTRU NIST submission in [Daniel J. Bernstein et al. 2024].

5. Conclusion

The experimental results indicate that the approach using the new lattice developed based on May's technique is promising for extending the feasibility of key recovery attacks on the ETRU system to larger dimensions.

With this new lattice, we were able to find the private key in some of the simulations conducted for a dimension of $n = 61$, which was not possible using the current lattice reduction techniques on the original lattice. This suggests that the dimension reduction of the lattice can be a valid attack strategy, paving the way for future investigations and improvements.

However, it is important to note that the use of a larger BKZ block resulted in a significant reduction in the algorithm's running time, and despite the observed improvements, we encountered limitations imposed by the scalability of the BKZ algorithm in larger dimensions.

Therefore, for future directions, it would be interesting to explore how advanced basis reduction techniques, such as the BKZ algorithm, can be adapted to take full advantage of this modified lattice. Additionally, the development of an algebraic BKZ using (see [Lyu et al. 2020]) the new lattice could be a promising area for future research.

References

Chen, Y. and Nguyen, P. Q. (2011). Bkz 2.0: Better lattice security estimates. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 1–20. Springer.

Daniel J. Bernstein, Tanja Lange, Chitchanok Chuengsatiansup, and Peter Schwabe (Accessed: 2024). NTRU Prime. `https://ntruprime.cr.yp.to`. Website.

Diffie W, H. M. (1976). New directions in cryptography. *In IEEE Transactions on Information Theory*, 22:644–654.

Gama, N. and Nguyen, P. Q. (2007). New chosen-ciphertext attacks on ntru. In *International Workshop on Public Key Cryptography*, pages 89–106. Springer.

Hoffstein, J., Pipher, J., and Silverman, J. H. (1998). Ntru: A ring-based public key cryptosystem. In *International algorithmic number theory symposium*, pages 267–288. Springer.

Howgrave-Graham, N., Nguyen, P. Q., Pointcheval, D., Proos, J., Silverman, J. H., Singer, A., and Whyte, W. (2003). The impact of decryption failures on the security of ntru encryption. In *Annual International Cryptology Conference*, pages 226–246. Springer.

Jaulmes, É. and Joux, A. (2000). A chosen-ciphertext attack against ntru. In *Annual international cryptology conference*, pages 20–35. Springer.

Karbasi, A. H. and Atani, R. E. (2015). Iltru: An ntru-like public key cryptosystem over ideal lattices. *International Association for Cryptologic Research*, Cryptology ePrint Archive:549–558.

Katherine Jarvis, M. N. (2015). Etru: Ntru over the eisenstein integers. *Designs, Codes and Cryptography*, 74:219–242.

Lenstra, A. K., Lenstra, H. W., and Lovász, L. (1982). Factoring polynomials with rational coefficients. *Mathematische annalen*, 261:515–534.

Lyu, S., Porter, C., and Ling, C. (2020). Lattice reduction over imaginary quadratic fields. *IEEE Transactions on Signal Processing*, 68:6380–6393.

Lyubashevsky, V. and Micciancio, D. (2009). On bounded distance decoding, unique shortest vectors, and the minimum distance problem. *Lecture Notes in Computer Science*, 5677:450–461.

May, A. (1999). Cryptanalysis of ntru. *unpublished*.

May A, S. J. (2001). Dimension reduction methods for convolutional modular lattices. *Lecture Notes in Computer Science*, 2146:110–125.

Monica Nevins, Camelia KarimianPour, A. M. (2010). Ntru over rings beyond z. *Designs, Codes and Cryptography*, 56:65–78.

Neal, K. (1987). Elliptic curves cryptosystems. *Mathematics of Computation*, 48:203–209.

Nguyen, P. Q. (2010). Hermite's constant and lattice algorithms, the lll algorithm: Survey's and applications. *Information Security and Cryptography*, pp:16–69.

NIST. Post-Quantum Cryptography Standardization. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization.

NIST (2019). Digital Signature Standard (DSS). https://csrc.nist.gov/pubs/fips/186-5/ipd.

Regev, O. (2009). On lattices, learning with errors, random linear codes, and cryptography. *Journal of ACM*, 56 no. 6:no. 6.

Rivest RL., Shamir A., A. L. (1978). A method for obtaining digital signatures and public key cryptosystem. *Communications of the ACM*, 21:120–126.

S. Lyu, C. P. and Ling, C. (2020). Lattice reduction over imaginary quadratic fields. *IEEE Transactions on Signal Processing*, 68:6380–6393.

Shor, P. (1994). Algorithms for quantum computation: Discrete logarithms and factoring. *Proceedings 35th Annual Symposium on Foundations of Computer Science*, page 124–134.

Sonika Singh, S. P. (2016). Generalizations of the ntru cryptosystem. 9:4823–6411.

Zhu, Z. and Tian, F. (2021). Comparison and intelligent analysis of ntru and etru signature algorithms for public key digital signature. *Journal of Physics: Conference Series*, 2083 no. 4:42009.

Lattice Basis Reduction Attack on Matrix NTRU

Thiago do Rêgo Sousa[1], **Tertuliano Souza Neto**[1]

[1]CEPESC, Brasília - DF

Abstract. *NTRU is one of the most important post-quantum cryptosystems nowadays and since its introduction several variants have been proposed in the literature. In particular, the Matrix NTRU is a variant which replaces the NTRU polynomials by integer matrices. In this work, we develop a lattice-based reduction attack on the Matrix NTRU cryptosystem that allows us to recover the plaintext. We also show that this system is completely vulnerable to the proposed attack for parameters that could be used in practice. In addition, we give sufficient conditions to avoid decryption failure for the Matrix NTRU.*

1. Introduction

Quantum computers are now a reality and if large-scale ones are built, they could break most of the public key cryptosystems used currently. That is one of the reasons that research on post-quantum systems (cryptographic systems underpinned on mathematical problems that are intractable by both quantum and conventional computers) has taken off in the last decades.

In 2016, the National Institute of Standards and Technology (NIST) started a public process for the standardization of post quantum algorithms that could be used to interoperate with existing communications protocols and networks currently used. One of the main mathematical problem underpinning such system is the shortest vector problem (SVP), which aims at finding a shortest vector in a structure called lattice (see [Silverman et al. 2008]).Two lattice-based systems have progressed significantly in the NIST Post-Quantum Cryptography standardization process [NIST]: the NTRU [Chen et al. 2020] and the NTRU Prime systems [Daniel J. Bernstein et al. 2024], both based on the original NTRU system from [Hoffstein et al. 1998].

The NTRU system was presented in 1996 to the cryptographic community during the CRYPTO 96 rump session. Since its introduction, the NTRU system has been extensively analyzed and extended in many different ways. Interesting review articles containing a wealth of examples and references include [Singh and Padhye 2016, Salleh and Kamarulhaili 2020] and [Mittal and Ramkumar 2022]. NTRU is based on modular algebra of polynomials in specific rings and such generalizations range from changing the polynomial coefficients to using matrix structures.

The first NTRU generalization using matrices was introduced in [Coglianese and Goi 2005], where key generation, encryption and decryption operate over matrices whose entries are polynomials. In 2008, [Nayak et al. 2008] proposed a matrix-based system as a variant of NTRU using only matrices with integer entries, replacing the arithmetic of truncated polynomials with a simple modular arithmetic on matrices. This new system, called matrix NTRU, was subject to some further analysis and even suggested for real data applications. Indeed, [Kumar et al. 2013] presented a framework for deploying matrix NTRU in real data applications.

Since its introduction the matrix NTRU has attracted some attention. A comparative study regarding speed and key sizes was performed in [Nayak et al. 2012] showing that matrix NTRU was faster than the classical NTRU for short dimensions. In addition, [Nayak et al. 2012] argue that matrix NTRU was more secure than the classical NTRU since matrices are noncommutative. However, we will show that the matrix NTRU system is seriously affected by the lattice attack developed in Section 5. This demonstrates that the matrix NTRU is far weaker than the classical NTRU system for comparable key sizes, contrary to what was accredited in the literature (see [Nayak et al. 2012] and [Kumar et al. 2013]).

Subsequent works on the matrix NTRU focused on expanding the key space and on eliminating restrictions on the matrices representing the public key by using Gram-Schmidt orthogonalization and companion matrices ([Luo and Lin 2011, Tripathi et al. 2016, Mamdikar et al. 2018]).

With regards to the system's security, [Nayak et al. 2011] applied a reaction type attack to matrix NTRU, by adapting the results of the original attack presented in [Hall et al. 1999] for the NTRU system. In addition, a meet in the middle attack was recently presented in [Wijayanti et al. 2023]. However, the simulation studies presented in these works only show the performance of the aforementioned attacks for very low dimensions of the matrix NTRU, which cannot be used in practice.

Our main contribution to matrix NTRU is in developing a lattice-based attack and analyse its resistance in dimensions much higher than the ones suggested for practical applications in [Kumar et al. 2013]. We will show in Section 5 how this attack can recover the private key (up to a permutation). In addition, this attack is further used to construct a message recovery attack that works for a dimension of the system of up to 110. According to [Nayak et al. 2012], a matrix NTRU system such as this is equivalent to an NTRU system using polynomials of degree 110^2, which is not even close to being broken by lattice-based techniques ([Chen et al. 2020]).

In addition to the lattice attack, we also analyse the decryption failure rate of the matrix NTRU. The results of Proposition 2 give sufficient conditions that depend only on the system's public parameters to avoid decryption failure.

The rest of the paper is organized as follows. In Section 2, we describe the NTRU cryptosystem in details and a brief description of Matrix NTRU is given in Section 3. Sufficient conditions to avoid decryption failure for the Matrix NTRU are also presented in this section. Then, in Section 4, we construct the lattice structure associated with the matrix NTRU and report the results of an attack to recover the private key. This attack is further refined in Section 5 to construct a message recovery type attack that works for a dimension of up to $n^2 = 110^2$ on a personal computer. The results of the previous section are summarized in Section 6 where we explain why the Matrix NTRU is far weaker than the NTRU cryptosystem concerning the lattice attack.

2. The NTRU Cryptosystem

Let n and p be prime numbers. Let $q \geq 1$ be an integer such that $(n, q) = (p, q) = 1$. The main arithmetic operations in the NTRU cryptosystem are calculations over polynomials defined over the rings R, R_p e R_q defined as:

$$R = \frac{\mathbb{Z}[x]}{(x^n - 1)}, R_p = \frac{(\mathbb{Z}/p\mathbb{Z})[x]}{(x^n - 1)}, R_q = \frac{(\mathbb{Z}/q\mathbb{Z})[x]}{(x^n - 1)}.$$

We can notice that the ring R is related to the other two. In fact, for every $a(x)$ in R, we can identify it to an element in R_p or R_q by applying reduction modulo p or q, respectively, over the coefficients of $a(x)$. We say that $a(x)$ is a ternary polynomial if its coefficients lie in the set $\{-1, 0, 1\}$.

The NTRU cryptosystem works as follows.

1. Key Generation: Alice chooses two ternary polynomials at random, namely, $f(x)$ and $g(x)$. Next, Alice tries to compute the inverse of f over the rings R_q, R_p, i.e, $f_q(x) = f(x)^{-1} \in R_q$ and $f_p(x) = f(x)^{-1} \in R_p$ until she succeeds. Finally, she computes the polynomial

   $$h(x) = f_q(x) * g(x) \in R_q,$$

   where $*$ denotes polynomial multiplication in R_q, i.e., a cyclic convolution product as defined in [Hoffstein et al. 1998, Section 1.1] The polynomial $h(x)$ is Alice's public key and the pair $(f(x), f_p(x))$ is her private key.
2. Encryption: Suppose Bob wants to send an encrypted message to Alice and let $m(x) \in R_p$ be Bob's plaintext. Next, Bob chooses, at random, a ternary polynomial $r(x)$ and computes

   $$e(x) \equiv ph(x) * r(x) + m(x) \mod q.$$

   Notice that Bob's ciphertext $e(x)$ is an element of the ring R_q.
3. Decryption: Once Alice receives Bob's ciphertext $e(x)$, she starts the decryption process by computing

   $$a(x) \equiv f(x) * e(x) \mod q.$$

   Then the reduction modulo p gives the desired plaintext

   $$b(x) \equiv f_p(x) * a(x) \mod p.$$

Because of the randomness of the polynomial $r(x)$, NTRU is a probabilistic cryptosystem, since a message $m(x)$ can be encrypted to several ciphertexts $ph(x) * r(x) + m(x)$, depending on each instance of $r(x)$. However, in doing so, we can introduce a vulnerability into the NTRU cryptosystem, since some ciphertext may not decrypt correctly to the original message, a phenomenon known as a decryption failure. Some attacks in the literature take advantage of these decryption failures [Howgrave-Graham et al. 2003, Gama and Nguyen 2007, Jaulmes and Joux 2000] and therefore we should choose the parameters carefully.

3. The Matrix NTRU Cryptosystem

Let p be a prime number and $q >> p$ an integer such that $(p, q) = 1$. Let

$$A = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix}$$

be an $n \times n$ matrix whose entries are integer numbers, that is, $A \in M_n(\mathbb{Z})$. In a similar way we have done before with polynomials, we say that A is a ternary matrix if all of its entries lie in the set $\{-1, 0-, 1\}$.

We say that A is reduced modulo p, denoted as $A \mod p$, if every entry of A is reduced modulo p. Therefore

$$A \mod p = \begin{pmatrix} a_{11} \mod p & \dots & a_{1n} \mod p \\ \vdots & \ddots & \vdots \\ a_{n1} \mod p & \dots & a_{nn} \mod p \end{pmatrix}.$$

Notice that, in the latter case, we can view matrix A as an element of the ring $M_n(\mathbb{F}_p)$. In this work, we are going to consider the center lift operation modulo p and q in the decryption process. In that case, it can be useful to consider a noncanonical representation of the elements on the rings. Therefore, when we are dealing with a prime modulus, we have

$$\mathbb{F}_p = \left\{ -\frac{p-1}{2}, \dots, \frac{p-1}{2} \right\}$$

and for a composite number, we have

$$\mathbb{Z}_q = \left\{ -\frac{q}{2} + 1, \dots, \frac{q}{2} \right\}.$$

Modular arithmetic over the rings $M_n(\mathbb{F}_p)$ and $M_n(\mathbb{Z}_q)$ is what underpins the Matrix NTRU system. In such rings we can add, subtract and multiply matrices in the same way we have done with matrices over the field of real numbers. However, in order to invert a matrix in $M_n(\mathbb{F}_p)$ we have to be sure that p and the determinant of A are relatively prime [Jacques-García et al. 2022, Wijayanti et al. 2023]. If that is the case, then there exists a (unique) matrix B such that $AB = BA = I$ in $M_n(\mathbb{F}_p)$, where I is the identity matrix. The matrix B is called the inverse of A modulo p and denoted $A^{-1} \mod p$.

The matrix NTRU system works as follows.

1. Key generation: In the Matrix NTRU cryptosystem, the private and public keys are $n \times n$ matrices. First, we choose a pair of ternary matrices F, G such that $F \mod p$ and $F \mod q$ are invertible in $M_n(\mathbb{F}_p)$ and $M_n(\mathbb{Z}_q)$, respectively. Then we compute the matrices F_p and F_q, where $F_p = F^{-1} \mod p$ and $F_q = F^{-1} \mod q$. The pair F, F_p is the private key and the parameters F, G, F_p, F_q should be kept in secret. Now, we can compute the public key by performing the calculation

$$H = pF_qG \mod q.$$

2. Encryption:To encrypt a message, we encode it as a matrix $M \in M_n(\mathbb{F}_p)$ and choose a ternary matrix $R \in M_n(\mathbb{Z})$, at random. Now, we compute the ciphertext as

$$E = HR + M \mod q.$$

3. Decryption: To decrypt, we compute

$$A = FE \mod q,$$

where F is the private key we saw previously. Next, we reduce A modulo p and compute

$$B = F_p A \mod p.$$

If everything went well, the matrix B will be precisely the message M we started with before the application of the encryption function.

However, depending on the parameters selection, we can have errors when operating on Matrix NTRU, just as we have when dealing with NTRU cryptosystem. Sometimes, the matrices B and M can be different and we say there is a decryption failure if that is the case. We notice that the authors [Nayak et al. 2008] did not address that question. However, it turns out that this is an important issue concerning any probabilistic cryptosystem. To fill such a lack in their proposed algorithm, we prove the following result, which shows that there are parameter selections to prevent failure decryption on Matrix NTRU.

Proposition 1. *Suppose that p, q and n are fixed parameters for the Matrix NTRU defined above. If $n(3p-1) < q$, then any ciphertext E resulting from the encryption of a message M with private key F, decrypts correctly to M.*

Proof. During decryption, one computes

$$\begin{aligned} A &= FE \mod q \\ &= F(HR + M) \mod q \\ &= FHR + FM \mod q \\ &= pFF_qGR + FM \mod q \\ &= pGR + FM \mod q. \end{aligned} \tag{1}$$

Following the same arguments as in the proof of Proposition 6.48 in [Silverman et al. 2008], we need to bound the largest coefficient (in modulus) of each entry of the matrix $pGR + FM$ computed exactly (without module q). For decryption to work, the above matrix should have entries whose absolute value does not exceed $q/2$. Since every entry of the matrices G and R lie in $\{-1, 0, 1\}$, the largest possible entry of the product GR is upper bounded by n. On the other hand, the matrix M has coefficients between $-(p-1)/2$ and $(p-1)/2$ and F has coefficients lying in $\{-1, 0, 1\}$. Therefore, all coefficients of the product FM can be bounded by $n(p-1)/2$. Finally, the entries of the matrix $pGR + FM$ are all upper bounded in module by $np + n(p-1)/2 = n(3p-1)/2$. Under the assumption of the proposition, this can be further bounded by $q/2$ which will ensure correct decryption of the ciphertext. □

We implemented the Matrix NTRU in [Team 2024] and assessed the decryption failure rate for a set of different parameters. Table 1 shows the results:

4. Lattice attack on the private key

In the following section we show how the problem of finding the private key F of the matrix NTRU system is connected to solving a well-known problem in lattices, namely finding the shortest vector in a lattice. A lattice can be defined in the following way.

Table 1. Decryption failure estimated probability for the Matrix NTRU system for fixed $p = 3$ and varying n and q. For each parameter setting a new key and message were generated and encrypted. Results present the proportion of messages decrypted correctly over 10000 repetitions.

	q										
n	32	64	79	128	256	307	512	701	1024	2048	4096
5	0.008	0	0	0	0	0	0	0	0	0	0
10	0.775	0	0	0	0	0	0	0	0	0	0
20	1	0.210	0.008	0	0	0	0	0	0	0	0
30	1	0.989	0.397	0	0	0	0	0	0	0	0
40	1	1	0.989	0.002	0	0	0	0	0	0	0
50	1	1	1	0.039	0	0	0	0	0	0	0
60	1	1	1	0.257	0	0	0	0	0	0	0
70	1	1	1	0.734	0	0	0	0	0	0	0
80	1	1	1	0.984	0	0	0	0	0	0	0
90	1	1	1	1	0	0	0	0	0	0	0
100	1	1	1	1	0	0	0	0	0	0	0
110	1	1	1	1	0	0	0	0	0	0	0

Definition 1. ([Silverman et al. 2008, Section 6.4]) Let $f_1, \ldots, f_n \in \mathbb{R}^n$ be a set of linearly independent vectors. The lattice L generated by $f_1, \ldots, f_n$ is the set of linear combinations of $f_1, \ldots, f_n$ with coefficients in $\mathbb{Z}$,

$$L = \{\alpha_1 f_1 + \cdots + \alpha_n f_n : \alpha_1, \ldots, \alpha_n \in \mathbb{Z}\}. \tag{2}$$

The set of vectors $f_1, \ldots, f_n$ is called the lattice basis and it is usual to stack them into a matrix and work with the matrix as being the lattice basis that generates L. A fundamental problem in lattice is finding a shortest nonzero vector in the lattice which minimizes the Euclidean norm $\|f\|$. This is called the shortest vector problem (SVP). It is important to notice that the SVP problem asks for a shortest vector and not the shortest vector since e.g. f and $-f$ have the same Euclidean norm. According to [Silverman et al. 2008] Finding a solution for the SVP problem can be used to break various cryptosystems, in particular the NTRU system from Section 2 (for certain parameters) and as we will show next how it can be used to finding the private key F in the matrix NTRU system. It is worth noticing that the current version of NTRU submitted to NISTs post quantum competition [Chen et al. 2020] have parameters that avoid private key attacks with current computational resources and it is practical. On the other hand, the matrix NTRU variant has a serious vulnerability in its construction, that makes it breakable for quite high values of parameters that could be used in practice.

Proposition 2. *Let F, G be the private keys and let H be the public key of the matrix NTRU with parameters n, p, q. Let (f_k, g_k) be the k-th line of F and G respectively and p^{-1} be the inverse of p module q. Then, (f_k, g_k) belongs to the lattice generated by the lines of the $2n \times 2n$ block matrix*

$$L = \begin{pmatrix} I_n & p^{-1}H \\ 0_n & qI_n \end{pmatrix}, \tag{3}$$

where I_n is the n dimensional identity matrix and 0_n is the n dimensional zero matrix. In other words, the unknown vector (f_k, g_k) can be written as an integer linear combination of the lines of L (which has only known quantities).

Proof. The public key is defined as $H = pF_qG$, where p and q are coprime, and F and G are the private keys created during the key generation step. Multiplying both sides of the above equation by Fp^{-1} we get $Fp^{-1}H = G \mod q$. This implies that there exists a matrix $A \in M(\mathbb{Z})$ such that

$$G = F(p^{-1}H) + qA, \tag{4}$$

Let $V = (\alpha H)$ and denote by f_k, g_k and a_k the k-th line of the matrices F, G and A. Equation (4) implies that $g_k = f_k V + qa_k$. Therefore, the $1 \times 2n$ vector (f_k, g_k) can be written as

$$\begin{aligned}
(f_k, g_k) &= (f_k, f_k V + a_k(qI_n)) \\
&= (f_k I_n + a_k 0_n, f_k V + a_k(qI_n)) \\
&= f_k(I_n, V) + a_k(0_n, qI_n) \\
&= f_{k1}(1, 0, \ldots, 0, 0, v_{11}, v_{12}, \ldots, v_{1n}) \\
&+ f_{k2}(0, 1, \ldots, 0, 0, v_{21}, v_{22}, \ldots, v_{2n}) \\
&\ldots \\
&+ f_{kn}(0, 0, \ldots, 0, 1, v_{n1}, v_{n2}, \ldots, v_{nn}) \\
&+ a_{k1}(0, 0, \ldots, 0, 0, q, 0, \ldots, 0) \\
&+ a_{k2}(0, 0, \ldots, 0, 0, 0, q, \ldots, 0) \\
&\ldots \\
&+ a_{kn}(0, 0, \ldots, 0, 0, 0, 0, \ldots, q).
\end{aligned} \tag{5}$$

Since all $1 \times 2n$ dimensional vectors at the rhs of (5) equation are exactly the lines of the matrix L in (3), the proof is completed. □

Recall that the matrices F and G have only coefficients in $\{-1, 0, 1\}$, and therefore any line of the type (f_k, g_k) is a relatively short vector in the lattice L for large q. Indeed, using a Gaussian heuristic, the shortest vector expected from a lattice L depends only on the dimension of L and its determinant. Since L is upper triangular, we have $\det(L) = q^n$. Therefore, the length of the shortest vector expected from L is

$$l = \sqrt{\frac{n}{2\pi e}}(\det(L))^{1/(2n)} = \sqrt{\frac{nq}{2\pi e}}.$$

The target vector (f_k, g_k) has varying norm depending on how the private key is chosen. If every entry is chosen with uniform probability on $\{-1, 0, 1\}$, then, its expected norm is $\alpha = \sqrt{2n/3}$. This means that for higher values of q, the target vector has norm smaller than the expected by the by the Gaussian heuristics, which means that the LLL algorithm has a high probability of find a short vector in L (see [Silverman et al. 2008]).

Another interesting fact is that we do not need to attack the whole private key F which has dimension $2n$. Using a suitable algorithm to solve the SVP in L we can try to find any line of F and hopefully all lines separately reducing the complexity of the attack

by a factor of n instead of n^2 (when the attacker is interested in obtaining all entries of F at once).

Solving the SVP problem can be done in many ways, and one commonly done is by using lattice-based reduction algorithms such as the famous LLL algorithm from [Lenstra et al. 1982] and or the BKZ algorithms [Chen and Nguyen 2011]. For more details on the LLL algorithm see e.g. [Bremner 2011, Chapter 4]. These base reduction algorithms try to find a shorter basis for the lattice L, and in doing so, they usually return potential short vectors for the lattice L and we can test if they correspond to any line of the matrix F.

In addition to using the LLL algorithm, there is an improved variant of lattice based reduction algorithms called BKZ as defined in [Chen and Nguyen 2011]. The BKZ algorithm, proceeds with repeated local improvements to the lattice basis. One of its simple implementations is a recurring set of calls to SVP oracle solvers of dimension set by the block size parameter of the BKZ algorithm and LLL calls and will be used in the following for recovering the private key. Although these algorithms can be further refined to lead to better solutions for the SVP problem, the first vectors of the BKZ output are already short enough to give us candidates for lines of the private key matrix F.

In what follows we use the result of Proposition 2 to recover the private key matrix F in the following way.

Algorithm 1: Recovering private key F:

Let H the public key of the matrix NTRU system with parameters n, p, q.

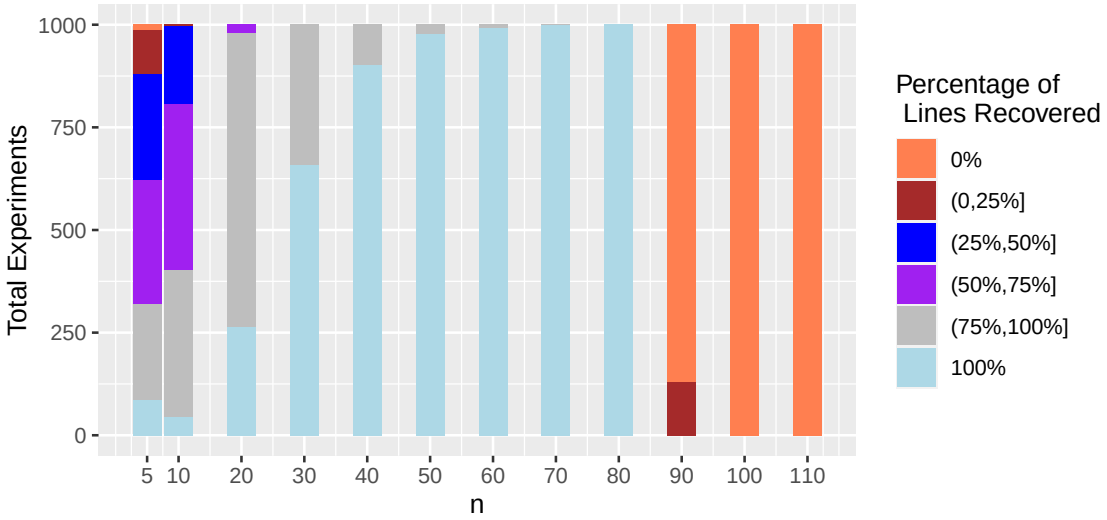
1. Compute $p^{-1} \mod q$and create the matrix L from eq.(3).
2. Run the BKZ algorithm on L and get L_{red}.
3. Use L_{red} to create a submatrix L^*_{red} with entries corresponding to the first n columns of L_{red}.
4. For each line f of the matrix F, check whether or not f is contained in one of the lines of L^*_{red}.

We implemented the above algorithm to assess the performance of the attack for several values of the parameters. Namely, we choose $q \in \{256, 4096\}$ for varying n and recorded what proportion of the lines of F one can recover from the attack using only the public key and public parameters p, q, n.

We see that the attack, in almost all experiments, can recover all lines of the matrix F for $n \in 40, 50, \ldots, 80$ and $q = 256$ and for $n \in 40, 50, \ldots, 110$ and $q = 4096$. For the case $q = 4096$ we also tried $n = 113$ and obtained success for quite some experiments, but in some cases the BKZ algorithm terminates with the message `terminate called recursively` and does not return the reduced basis.. At this stage, the lattice dimension is already $2n = 226$ and it gets more difficult to reduce the basis. For all settings we set the block of the BKZ algorithm to 10 since this allowed quite good results. One could increase the block size at the expense of a bigger running time.

One issue with the attack of algorithm 1 is that even though we can recover some lines of the matrix F, we still do not know how to reorder them to reconstruct the true F and use it for decryption. Nevertheless, we will see in the next Session that any permutation of lines of the private key F can be used to successfully decrypt a message encrypted

Figure 1. Performance of the attack of algorithm 1 for recovering lines of the matrix F for the matrix NTRU with parameters n and fixed $q = 256$ (upper figure) and $q = 4096$ (bellow figure). For each experiment, we generate new keys F and H and apply the attack recording how many experiments were able to recover all lines of F (100%), between 1 and $n - 1$ lines of F and none of the lines of F (0%).



with F and this allows us to construct a message recovery attack on the matrix NTRU system.

5. Message recovery attack

In the previous section we showed how to construct the associated lattice for the matrix NTRU system and demonstrated the viability of the attack for recovering, up to a permutation, the whole private key matrix F. In what follows we show that this attack can be used to successfully decrypt a message encrypted with F, even though the correct order of the lines of F is not known by the attacker.

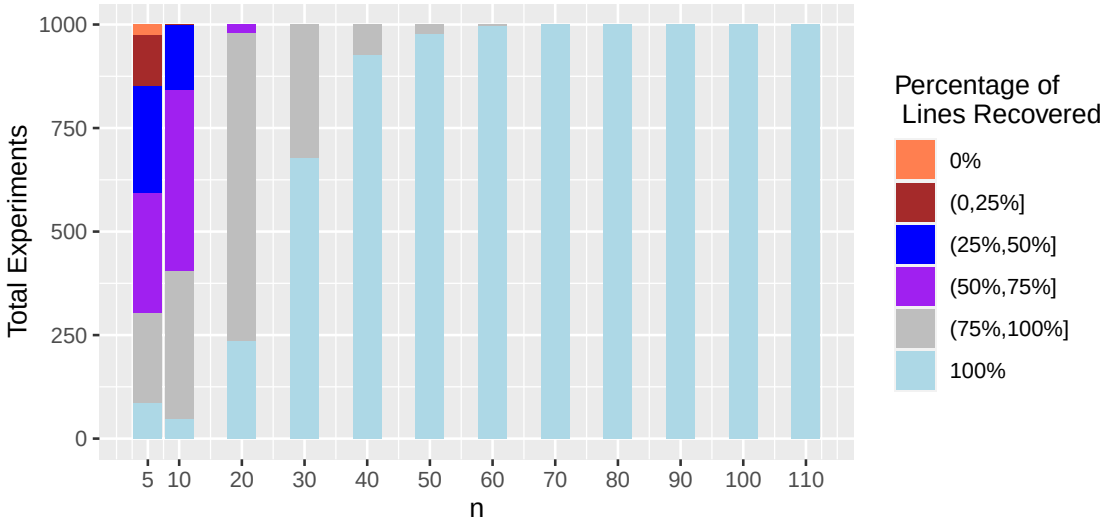
Proposition 3. *Let F, H be a corresponding private and public key pair for the matrix NTRU system with parameters n, p, q and let E be a ciphertext associated with the encryption of a message M such that the decryption process of E using F correctly returns M. Let F^* be a matrix formed by permuting the lines of the matrix F. Then, applying the decryption process to E using F^* returns the message M.*

Proof. By the construction of F^*, there exists an unimodular matrix D_1, such that $F^* = D_1F = FD_2$. Applying the decryption process gives us

$$A_2 = F_2E \mod q = D_1F(HR + M) \mod q.$$

Using the definition of the matrix H and the fact that $FF_q = I$ we get,

Figure 2. Same settings as in Figure 3 but with $q = 4096$.



$$\begin{aligned} A_2 &= D_1F(HR + M) \mod q \\ &= D_1F(pF_qGR + M) \mod q \\ &= D_1pFF_qGR + D_1FM \mod q \\ &= pD_1GR + D_1FM \mod q \end{aligned} \tag{6}$$

Next we are going to make this equations modulo p and notice that since D_1 only change signs and permute lines of GR, we can still make this operation without incurring in the risk of extrapolating the norm of the entries of the matrix D_1GR. Therefore, we have

$$A_2 \mod p = 0 + D_1FM = F^*M.$$

Finally, using the inverse of F_p^* of $F^* \mod p$ we compute

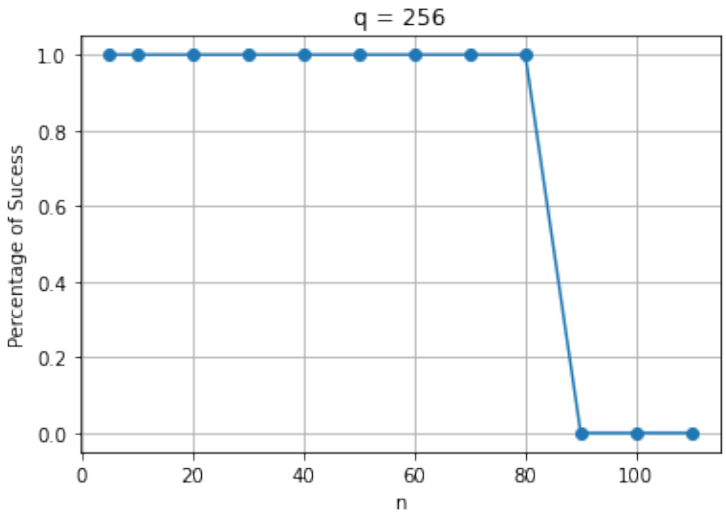
$$C_2 = F_p^*A_2 = F_p^*F^*M \mod p = M.$$

□

We use the result of Proposition 3 in combination with Algorithm 1 to create a practical message recovery attack. This attack uses the upper left part of the reduced lattice basis returned by the BKZ algorithm as a potential key. As we saw in Section 4, this gives us in several cases the whole private key (up to a permutation of lines). The result of Proposition 3 says this potential key can still be used to decrypt a message successfully and this is tested in practice.

For each experiment, we generate new key pair F, H a new message M, encrypt it and try to decrypt with the key F^* returned from the attack of algorithm 1. The experiment was repeated 100 times on an Intel(R) Core(TM) i5-9500T CPU 2.20GHz and the success rate reported in Figures 3 and 4. In the worst case scenario ($n = 110$ and

Figure 3. Performance of the attack of algorithm 1 for recovering lines of the matrix F for the matrix NTRU with parameters n and fixed $q = 256$ (upper figure) and $q = 4096$ (bellow figure). For each experiment, we generate new key pair F, H a new message M, encrypt it and try to decrypt with the key F^* returned from the attack of algorithm 1. The experiment was repeated 100 times and the sucess percentage recorded.

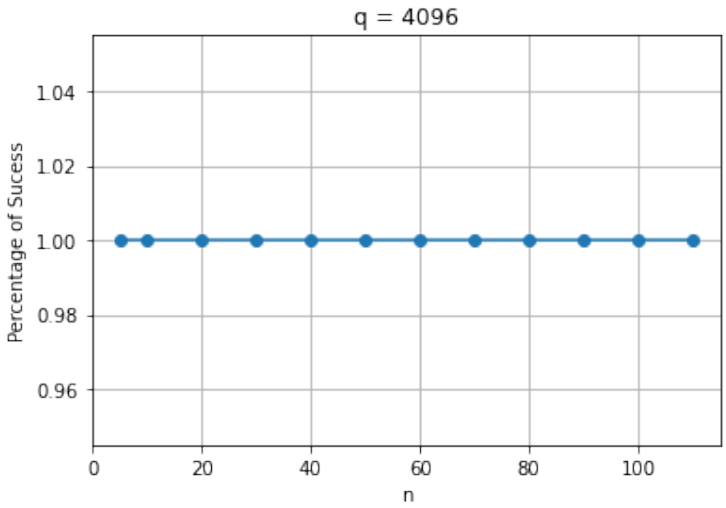


$q = 4096$) we can recover the true message in less than one minute. For $q = 256$ the attack works for dimension until $n = 80$ and for $q = 4096$ it works for dimensions up to 100. Since [Nayak et al. 2012] a matrix NTRU of dimension n with an NTRU of dimension n^2, this clearly shows that matrix NTRU is far weaker than NTRU since an NTRU with dimension $100^2 = 10000$ is far from being broken. In fact, the highest security suggested for NTRU in practical applications has parameters $n = 821$ and $q = 4096$ as shown in [Chen et al. 2020, Section 1.6]. By the results of this section, the matrix NTRU of dimension n^2, should be comparable (in terms of the lattice attack) to at most an NTRU of dimension n. Therefore, even though the Matrix NTRU can encode a similar number of bits in the private key as the NTRU, its internal structure makes it vulnerable to attacks in a much smaller dimension.

6. Conclusion

In [Nayak et al. 2012], the authors compare the speed performance for encryption and decryption of the classical NTRU and the Matrix NTRU arguing that matrix NTRU is faster than NTRU for comparable parameters. They compare a matrix NTRU with parameters n with an NTRU with parameter n^2, since the original idea of the matrix NTRU system is to encode the private key polynomial of degree n^2 into a matrix of dimension $n \times n$. In terms of brute force search for the private key they are comparable. In terms of lattice-based attacks, the post-quantum NIST finalist NTRU encrypt (REF) with parameters $n = 509$ and $q = 2048$ has already moderate security and cannot be broken by current the available techniques using lattice attacks. The equivalent matrix NTRU would have dimension around 23, an integer approximation to $\sqrt{509}$. However, a matrix NTRU with parameters $n = 23$ is completely vulnerable to lattice attacks on a personal computer as showed in Section 4. In fact we can attack such system for even higher dimension.

Figure 4. Same settings as in Figure 3 but with $q = 4096$.



The bottleneck of the proposed attack is in finding a suitable short vector for the matrix NTRU system using lattice basis reduction algorithms. Therefore, any improvement on these techniques such as replacing the BKZ routine by, e.g., the ones described in ([Albrecht and Ducas 2021], [May and Silverman 2001], and [Bi and Han 2021, Zhao and Ye 2023]) would improve the results showed in Figures 1 and 2.

This vulnerability of the matrix NTRU system is very serious since its security drooped from n^2 to n using the refereed lattice attacks developed here. In comparisons with the NTRU the matrix NTRU tries to creates a faster variant of NTRU by separating the private key into slices (lines of a matrices) and using this to create the public key. On the other hand this come at a very high security cost since just one line is used at a time to construct the lines of the public key. That means it runs faster at the cost of reducing the diffusion during the creation of the public key. On the other hand, NTRU creates public keys by using convolution of polynomials, and therefore, every coefficient contributes to create every coffined of the public key.

To conclude, we believe that the proposed attack can be adapted to other NTRU like systems relying on matrices operations such as the one based on the Gaussian Integers.

References

Albrecht, M. and Ducas, L. (2021). Lattice attacks on ntru and lwe: a history of refinements. *Cryptology ePrint Archive*.

Bi, J. and Han, L. (2021). Lattice attacks on ntru revisited. *IEEE Access*, 9:66218–66222.

Bremner, M. (2011). *Lattice basis reduction*. CRC Press New York.

Chen, C., Danba, O., Hoffstein, J., Hulsing, A., Rijneveld, J., Schanck, J. M., Schwabe, P., Whyte, W., and Zhang, Z. (2020). Ntru: algorithm specifications and supporting documentation (2019). *URL: https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.*, 1.

Chen, Y. and Nguyen, P. Q. (2011). Bkz 2.0: Better lattice security estimates. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 1–20. Springer.

Coglianese, M. and Goi, B.-M. (2005). Matru: A new ntru-based cryptosystem. In *Progress in Cryptology-INDOCRYPT 2005: 6th International Conference on Cryptology in India, Bangalore, India, December 10-12, 2005. Proceedings 6*, pages 232–243. Springer.

Daniel J. Bernstein, Tanja Lange, Chitchanok Chuengsatiansup, and Peter Schwabe (Accessed: 2024). NTRU Prime. `https://ntruprime.cr.yp.to`. Website.

Gama, N. and Nguyen, P. Q. (2007). New chosen-ciphertext attacks on ntru. In *International Workshop on Public Key Cryptography*, pages 89–106. Springer.

Hall, C., Goldberg, I., and Schneier, B. (1999). Reaction attacks against several public-key cryptosystem. In *Information and Communication Security: Second International Conference, ICICS'99, Sydney, Australia, November 9-11, 1999. Proceedings 2*, pages 2–12. Springer.

Hoffstein, J., Pipher, J., and Silverman, J. H. (1998). Ntru: A ring-based public key cryptosystem. In *International algorithmic number theory symposium*, pages 267–288. Springer.

Howgrave-Graham, N., Nguyen, P. Q., Pointcheval, D., Proos, J., Silverman, J. H., Singer, A., and Whyte, W. (2003). The impact of decryption failures on the security of ntru encryption. In *Annual International Cryptology Conference*, pages 226–246. Springer.

Jacques-García, F. A., Uribe-Mejía, D., Macías-Bobadilla, G., and Chaparro-Sánchez, R. (2022). On modular inverse matrices a computation approach. *South Florida Journal of Development*, 3(3):3100–3111.

Jaulmes, É. and Joux, A. (2000). A chosen-ciphertext attack against ntru. In *Annual international cryptology conference*, pages 20–35. Springer.

Kumar, V., Mamdikar, M. R., and Gosh, D. (2013). Matrix formulation of ntru algorithm using multiple public keys from matrix data bank for high degree polynomials. In *CEEE*, pages 191–198.

Lenstra, A. K., Lenstra, H. W., and Lovász, L. (1982). Factoring polynomials with rational coefficients. *Mathematische annalen*, 261:515–534.

Luo, X.-R. and Lin, C.-H. J. (2011). Discussion on matrix ntru. *IJCSNS International Journal of Computer Science and Network Security*, 11(1):32–35.

Mamdikar, M. R., Kumar, V., and Ghosh, D. (2018). Implementation of automatic invertible matrix mechanism in ntru matrix formulation algorithm.

May, A. and Silverman, J. H. (2001). Dimension reduction methods for convolution modular lattices. In *International Cryptography and Lattices Conference*, pages 110–125. Springer.

Mittal, S. and Ramkumar, K. (2022). A retrospective study on ntru cryptosystem. In *AIP Conference Proceedings*, volume 2451. AIP Publishing.

Nayak, R., Pradhan, J., and Sastry, C. (2011). Reaction attacks in the matrix scheme of ntru cryptosystem. In *International Conference on Advances in Information Technology and Mobile Communication*, pages 27–32. Springer.

Nayak, R., Pradhan, J., and Sastry, C. (2012). Evaluation of performance characteristics of polynomial based and lattice based nrtu cryptosystem.

Nayak, R., Sastry, C., and Pradhan, J. (2008). A matrix formulation for ntru cryptosystem. In *2008 16th IEEE International Conference on Networks*, pages 1–5. IEEE.

NIST. Post-Quantum Cryptography Standardization. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization.

Salleh, N. and Kamarulhaili, H. (2020). Ntru public-key cryptosystem and its variants: An overview. *Int. l J. of Cryptology Research*, 10(1):1–21.

Silverman, J. H., Pipher, J., and Hoffstein, J. (2008). *An introduction to mathematical cryptography*, volume 1. Springer.

Singh, S. and Padhye, S. (2016). Generalisations of ntru cryptosystem. *Security and Communication Networks*, 9(18):6315–6334.

Team, S. D. (2024). *SageMath*. Available from `https://www.sagemath.org`.

Tripathi, B., Thakur, K., Nayak, R., Sastry, C., and Pradhan, J. (2016). Ntru cryptosystem with companion matrix. *A matrix formulation for NTRU cryptosystem*, pages 1–5.

Wijayanti, I. E. et al. (2023). The meet-in-the-middle attack on the matrix ntru cryptosystem. In *2023 IEEE International Conference on Cryptography, Informatics, and Cybersecurity (ICoCICs)*, pages 149–153. IEEE.

Zhao, Z. and Ye, Q. (2023). Revisiting lower dimension lattice attacks on ntru. *Information Processing Letters*, 181:106353.

Modified versions of ML-KEM based on Brazilian cryptographic resources

Vinícius Lagrota[1], Beatriz L. Azevedo[2], Mateus de L. Filomeno[2], Moisés V. Ribeiro[2]

[1] Research and Development Center for Communication Security (CEPESC)
Brasília, DF – Brazil.

[2]Electric Engineering Department
Federal University of Juiz de Fora (UFJF) – Juiz de Fora, MG – Brazil

{vinicius.lagrota, mateus.lima, mribeiro}@engenharia.ufjf.br,
beatriz.azevedo@estudante.ufjf.br

Abstract. *This paper outlines the Module-Lattice-Based Key-Encapsulation Mechanism Standard (ML-KEM) based on Brazilian cryptography to safeguard sensitive information. In this sense, it details two Brazilian cryptographic algorithms, Forró and Xote, and discusses the modifications in the ML-KEM to enable their use as symmetric primitives. Relying on experimental results regarding execution time, we show that ML-KEM with Xote surpasses ML-KEM with SHAKE or Forró while maintaining an equivalent level of security in tasks such as key pair generation, encapsulation, and decapsulation.*

1. Introduction

Cryptography is a cornerstone in safeguarding national sovereignty, sensitive information, and critical infrastructure against cyber threats and espionage activities, which is crucial for maintaining trust and security in an interconnected world. By ensuring the confidentiality, integrity, and authenticity of government communications, cryptographic measures protect against foreign actions that may introduce vulnerabilities and aid compliance with regulatory standards and international agreements, bolstering trust among citizens and stakeholders in handling sensitive data and communications. Strong cryptographic schemes are vital for governments to maintain national security, protect digital sovereignty, and cultivate a secure atmosphere for citizens, businesses, and governmental activities in the face of evolving cyber threats.

Cryptography, on the other hand, faces significant challenges with the rapid advancement of quantum computing technology. Once a cryptographically relevant quantum computer (CRQC) emerges, Shor's algorithm [Shor 1994] will render public-key cryptography vulnerable due to its ability to efficiently solve problems like factoring large numbers and discrete logarithms. In anticipation of the quantum era, National Institute for Standards and Technology (NIST) started a competition in 2017 to standardize Public Key Encryption (PKE)/Key Encapsulation Mechanism (KEM) and digital signature algorithms resistant to quantum (and classical) computer attacks, i.e., the post-quantum cryptography (PQC). So far, four algorithms have been standardized or selected to be. Among them, CRYSTALS-Kyber [Avanzi et al. 2021], which

uses Secure Hash Algorithm and Keccak (SHAKE) and Advanced Encryption Standard (AES) as symmetric primitives, is the only KEM. It has been standardized under the name FIPS-203 Module-Lattice-Based Key-Encapsulation Mechanism Standard (ML-KEM) [National Institute of Standards and Technology 2024].

As the ML-KEM underwent standardization by NIST, it garnered significant attention and research. As exemplified by AVX2 in its NIST submission, software optimizations have been explored, focusing on utilizing fast devices and intrinsic functions. For instance, [Wan et al. 2022] introduced a GPU-based implementation, whereas [Nguyen and Gaj 2021] proposed an ARMv8-targeted implementation leveraging NEON-based special instructions. Furthermore, hardware optimization strategies have also been investigated, including reconfigurable hardware implementations with side-channel protection [Jati et al. 2024], compact pure hardware implementations prioritizing performance and area efficiency [Xing and Li 2021], and the feasibility of ML-KEM on hardware-constrained devices [Costa et al. 2022], suggesting a system-on-a-chip (SoC) implementation for accelerating critical operations in hardware while managing the remainder operations in software.

A notable gap in the existing literature pertains to investigations into performance enhancements of ML-KEM using alternative symmetric primitives beyond SHAKE and AES. The use of other symmetric primitives is particularly relevant in cryptography; for instance, governments should use state-of-the-art encryption algorithms and protocols to safeguard sensitive information from unauthorized access and interception. In this regard, the use of PQC schemes such as the ML-KEM becomes imperative[1]. Nevertheless, each nation must develop its unique set of symmetric primitives to reduce dependence on third parties and mitigate external influence. In this regard, the Research and Development Center for Communication Security (CEPESC) has provided government cryptography for the Brazilian government since 1982. To enhance transparency, the CEPESC published a cryptographic library called *libharpia*, which was used in Brazilian elections [Pacheco et al. 2022]. Forró [Coutinho et al. 2022], a symmetric algorithm based on the add-rotate-XOR (ARX) architecture, was later proposed as a special public algorithm for Brazilian elections and, therefore, embedded in *libharpia*. However, it is important to highlight that it is not yet being used in the election process. Focusing on a slight structural modification of Forró, a fast implementation named Xote [Coutinho 2021] was introduced to reduce Forró's execution time. Integrating the Forró and Xote, cryptographic symmetric primitives, with ML-KEM is an interesting strategy to strengthen Brazilian national security and sovereignty.

To address this need, and based on an in-depth study of symmetric primitives in the ML-KEM, this paper introduces two modified versions of ML-KEM. The first modified version replaces SHAKE with Forró, while the second replaces SHAKE with Xote. Replacing SHAKE with Forró or Xote involves the use of modified core functions — i.e., pseudo-random function (PRF), extendable-output function (XOF), and key-derivation

[1]Regarding the use of PQC in government applications, National Cyber Security Centre (NCSC), a United Kingdom agency, and French Cybersecurity Agency (ANSSI) have reports in this matter, evaluating and discussing PQC [NCSC 2020, ANSSI 2022]. Nonetheless, as far as it is known, only the Federal Office for Information Security (BSI), a German federal agency, has officially recommended a PQC scheme [BSI 2020].

function (KDF) — including algorithm presentations, justifications, and integration in ML-KEM. Since the security level remains equivalent across ML-KEM, Forró-based ML-KEM, and Xote-based ML-KEM, we provide a comparison in terms of processing time. Experimental results indicate that the execution time for Forró-based ML-KEM is higher than that of ML-KEM (i.e., in between 3.63% and 6.42%), while Xote-based ML-KEM performs better than ML-KEM (i.e., in between 0.68% and 3.49%).

The remainder of this paper is organized as follows: Section 2 presents fundamental concepts of cryptography and the algorithms employed in this work. Section 3 delves into the implementation details, with a primary focus on the roles of Forró and Xote as PRF, XOF, and KDF within ML-KEM. The experimental results are discussed in Section 4, showcasing the execution time of the proposed methods. Finally, Section 5 offers concluding remarks and potential directions for future research.

1.1. General notation

The notation employed in this paper aligns with that used in the supporting documentation of ML-KEM [National Institute of Standards and Technology 2024]. Functions within the schemes operate on byte arrays as both input and output, where $\mathcal{B} = \{0, ..., 255\}$ represents unsigned 8-bit integers or bytes. Additionally, $\mathcal{B}^K$ denotes the set of byte arrays of K-length, and $\mathcal{B}^*$ represents byte arrays of any length (i.e., a byte stream). $\mathcal{U} = \mathcal{B}^4$ represents unsigned 32-bit integers. The symbol $||$ denotes the concatenation of two-byte arrays. Given a byte array a and a non-negative integer k, $a[k]$ refers to the byte array starting at byte k of a (with indexing starting at zero). Matrices and vectors are denoted by uppercase and lowercase bold letters, respectively, and $\mathbf{A}^T$ refers to the transpose of matrix $\mathbf{A}$.

2. Fundamentals

This section presents brief descriptions of KEM, ML-KEM, and Forró and Xote algorithms that are necessary for understanding the modified version of the ML-KEM. Subsection 2.1 briefly describes the KEM, Subsection 2.2 overviews ML-KEM, and Subsection 2.3 details Forró and Xote algorithms.

2.1. Key Encapsulation Mechanism

Let us consider that Alice and Bob desire to communicate using a PQC scheme. To do so, public-key and symmetric cryptographic schemes should be properly used. A KEM is a public-key cryptography scheme used for exchanging a secret shared (i.e., a key for symmetric cryptography) between two parties. It is generally divided into three main functions: key pair generation, encapsulation, and decapsulation. Key pair generation, executed by Alice, is responsible for generating public and private keys. Alice securely stores the private key, whereas her public key is securely sent to Bob. In turn, Bob receives the public key generated by Alice. He obtains a shared secret, executes the encapsulation on it, and then returns a ciphertext to Alice. Finally, using the ciphertext and the private key, Alice executes the decapsulation, which returns the same shared secret achieved by Bob. After those operations, Alice and Bob hold the same shared secret, enabling them to communicate securely and efficiently with symmetric cryptography. Figure 1 shows a block diagram that illustrates how KEM (public-key cryptography) is employed in a cryptographic scheme.

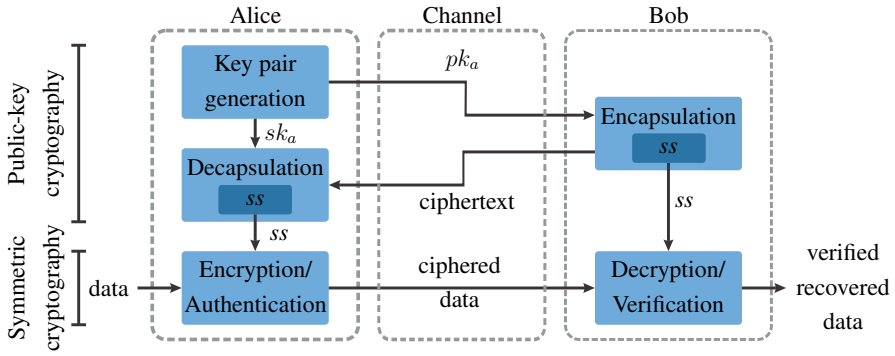


Figure 1. Block diagram illustrating the key agreement between parties and the encryption/decryption process. Note that pk_a, sk_a, and ss stand for public key, private key, and shared secret, respectively.

2.2. ML-KEM

The ML-KEM originated from the K-PKE via a slightly modified Fugisaki-Okamoto transform [Barbosa and Hülsing 2023]. It is the only PQC algorithm being standardized by NIST in the PKE/KEM category. Furthermore, the ML-KEM relies on the Module-Learning With Errors (M-LWE) problem [Langlois and Stehlé 2015, Albrecht and Deo 2017], which was designed to seek high performance without losing flexibility. The ML-KEM is available in three different versions, named ML-KEM-512, -768, and -1024, targeting distinct security levels and, consequently, ensuring a security level equivalent to AES-128, AES-192, and AES-256, respectively.

To run the ML-KEM, the following symmetric primitives are required: two hash functions, a XOF, a PRF, and a KDF. Moreover, to reduce code size and minimize vulnerability, ML-KEM opts for a single underlying symmetric primitive to fulfill all these functions. In this sense, ML-KEM defined SHAKE functions standardized in FIPS-202 [Dworkin 2015][2]. This standard also describes hash functions with the required output lengths and is designed to work as PRF and KDF. The symmetric primitives used by ML-KEM are detailed as follows:

- **Hash functions:** A hash function is an algorithm that takes input data and produces fixed-size output data. In the ML-KEM, the hash functions are instantiated as SHA3-256 and SHA3-512, as described in FIPS-202. The hash functions are required in key pair generation, encapsulation, and decapsulation processes.
- **Extended output function (XOF):** A XOF is a function that maps an arbitrary-length input bit string to an output bit string that can be extended to any desired length. The XOF instantiated by ML-KEM is the SHAKE-128, divided into two steps: absorbing and squeezing. The absorbing step has a seed and two distinct counters as input, which are absorbed by the sponge structure, modifying its internal state. On the other hand, the squeezing step outputs a bit string of the desired length based on the modified internal state provided by the absorbing step. The XOF is used as a step for generating two fundamental matrices for the ML-KEM,

[2]ML-KEM also offers AES-256 and SHA-2 as symmetric primitives, as those primitives can be hardware accelerated on various platforms. As this work does not approach hardware acceleration, only the SHAKE family is addressed.

namely $\hat{\mathbf{A}}$ and $\hat{\mathbf{A}}^T$, which are indirectly part of the public key. It significantly impacts the total execution time of the ML-KEM [Da Costa et al. 2022]. The XOF is required in key pair generation, encapsulation, and decapsulation processes.

- **Pseudo-random function (PRF):** a PRF is a structure used in cryptography that emulates a random oracle. It usually takes a key and a nonce as inputs and then outputs the requested number of bytes. In the ML-KEM, the PRF instantiated is the SHAKE-256. It uses random coins as a key and a counter as a nonce. In the ML-KEM, the PRF generates part of the private key and error vector, which is relevant for the M-LWE problem as it ensures computational hardness and security. The PRFs is required in key pair generation, encapsulation, and decapsulation processes.
- **Key-derivation function (KDF):** A KDF is a cryptographic algorithm that transforms a secret key, password, or key material into a derived key. The SHAKE-256 is also used as KDF in the ML-KEM. It receives a key as input and returns a derived key, i.e., the shared secret. The KDF is required by encapsulation and decapsulation processes.

2.3. Forró and Xote

Forró [Coutinho et al. 2022] and Xote [Coutinho 2021] are two symmetric algorithms. Aiming to enhance security performance, both algorithms evolved from Salsa20 [Bernstein 2008] and ChaCha20 [Bernstein et al. 2008], also known as Salsa20/R and ChaCha20/R where the variable R indicates the number of rounds. To understand Forró and Xote, the following paragraphs describe their evolution, tracing the progression from Salsa20 through ChaCha20 and Forró to Xote.

The stream cipher Salsa20 consists of addition, rotation, and XOR operations on 32-bits words, characterizing an ARX architecture. One of its main characteristics is its efficiency in hardware and software. Salsa20 operates on a state of 64-bytes (or 512-bits), organized as a 4×4 matrix with 32-bits integers. Its state is initialized with a 256-bits key, a 64-bits nonce, a 64-bits counter[3], and 4 constants of 32-bits each. The state matrix is altered in each round by a quarter round (QR) function, which receives 4 of the 16 integers of the state matrix and updates them. A round comprises 4 (four) applications of the QR function receiving different inputs at each call. The output of the Salsa20 is then defined as the sum of the initial state with the resulting state after R rounds[4]. For more details regarding the matrix construction, constants, and QR function of Salsa20, see [Bernstein 2008]. ChaCha20 was proposed as an improvement of Salsa20. Although they present the same structure, modifications in the QR function were made in ChaCha20, enhancing security. For more details regarding security, matrix construction, constants, and QR function of ChaCha20, see [Bernstein et al. 2008].

Forró was designed to evolve ChaCha20 using a novel concept called Pollination [Coutinho et al. 2022]. While ChaCha20 improves diffusion between rounds compared to Salsa20, Forró aims to enhance diffusion within rounds when compared to ChaCha20. This enhancement in Forró is achieved because the QR functions of Forró were modified to be applied dependently between columns and diagonals, which is not

[3]The concatenation of nonce and counter is the initialization vector (IV): $iv := nonce||counter$

[4]As default, R is defined as 20. However, reduced-round variants with R equal to 8 (eight) and 12 have also been introduced.

made in ChaCha20. Consequently, obtaining more diffusion with fewer operations (or rounds) is possible. As a result, the QR function of Forró receives 5 (five) integers instead of 4 (four) and needs to perform fewer rounds, 14 against 20 in ChaCha. For comparison, the best distinguishers against 5 rounds of Salsa, ChaCha, and Forró are, respectively, 2^8, 2^{16}, and 2^{130}. Therefore, Forró can deliver the same security as Salsa20 and ChaCha20 in fewer rounds.

Subsequently, Xote was developed as an evolution of Forró. It maintains identical parameters to Forró, including its QR function but employing two state matrices instead of one. Upon initializing the first matrix, it is duplicated into the second one with an incremented counter. While doubling the computational operations needed to process the state matrix, this design choice allows Xote to generate twice as much keystream as Forró without doubling the processing time. As a result, Xote exhibits improved execution time compared to Forró while delivering the same security level at the cost of using more memory. In summary, with fewer operations, Forró and Xote achieve the same security level as ChaCha20. Consequently, Forró and Xote can perform faster on certain platforms, especially constrained devices with low processing power.

3. ML-KEM instantiating Forró and Xote

To adapt ML-KEM to use Forró or Xote instead of SHAKE, we introduce modified versions of XOF, PRF, and KDF. The hash functions do not use Forró or Xote to maintain a domain separation, as recommended in [Avanzi et al. 2021]. To do so, the algorithms of modified versions of XOF, PRF, and KDF use the following functions[5]: {Forro, Xote}.Keysetup(·), {Forro, Xote}.IVsetup(·), {Forro, Xote}.QR(·), and {Forro, Xote}.Encrypt(·), found in [Coutinho et al. 2022]. In addition, {Forro,Xote}.GenerateBytes(·) is a slight modification of {Forro,Xote}.Encrypt(·). In {Forro,Xote}.GenerateBytes(·), the output is not the input *xored* with the keystream as in {Forro,Xote}.Encrypt(·); instead, the output of {Forro,Xote}.GenerateBytes(·) is directly the keystream. SHAKE functions are also described in detail by FIPS-202 [Dworkin 2015].

Note that the security of ML-KEM with Forró and Xote will be very similar to that of ML-KEM with SHAKE because Forró and Xote are also secure [Coutinho et al. 2022]. In particular, there is a small improvement in security since the XOF in ML-KEM using Forró and Xote provides 256-bits of security. In contrast, XOF in ML-KEM with SHAKE uses SHAKE-128, which provides 128-bits of security. It is important to note that the remaining introduced algorithms maintain the same security level as those they replace.

In the sequel, Subsections 3.1 to 3.3 detailed the modified versions of XOF, PRF, and KDF based on both Forró and Xote, which are required for instantiating symmetric primitives within the ML-KEM.

3.1. Extended-output function

Algorithms 1 and 2 respectively implement the {Forro,Xote}.XOF-absorb(·) and {Forro,Xote}.XOF-squeeze(·) functions proposed by the authors. As discussed in Sub-

[5]From now on, we use a notation where multiple functions are grouped by listing their names within curly braces before the function name. For instance, {Forró, Xote}.Function(·) refers to both Forro.Function(·) and Xote.Function(·). If any additional function like SHAKE has to be included, we add them to the list: {SHAKE, Forró, Xote}.Function(·).

section 2.2, the XOF is used for generating two matrices, $\hat{\mathbf{A}}$ and $\hat{\mathbf{A}}^T$, which are required in key pair generation, encapsulation, and decapsulation algorithms. The absorb step has a 3-tuple (ρ, j, i) and a state matrix st as input. In Forró and Xote, ρ plays as key, whereas i and j will be the first and second 32-bits words of nonce. As the output of the absorbing step, a modified state matrix st is returned. On the other hand, the squeeze step consists of receiving the processed state matrix st by {Forro,Xote}.XOF-absorb($\cdot$) and the length N of the output required. In this sense, {Forro,Xote}.XOF-squeeze($\cdot$) only calls the function {Forro, Xote}.GenerateBytes($\cdot$), generating N-bytes as output.

Algorithm 1 {Forro, Xote}.XOF-absorb(st, ρ, i, j)

Input:
State matrix: $st \in \mathcal{U}^{4\times4}$
Seed: $\rho \in \mathcal{B}^{32}$
Nonce: $i, j \in \mathcal{B}^{4}$
Output:
State matrix: $st \in \mathcal{U}^{4\times4}$
Procedure:
$iv = i||j$
{Forro, Xote}.Keysetup(st, ρ)
{Forro, Xote}.IVsetup(st, iv)
{Forro, Xote}.QR(st)
Return:
State matrix: st

Algorithm 2 {Forro, Xote}.XOF-squeeze(st, N)

Input:
State matrix: $st \in \mathcal{U}^{4\times4}$
Output length: $N \in \mathcal{U}$
Output:
Byte string: $out \in \mathcal{B}^{*}$
Return:
Byte string: out := {Forro, Xote}.GenerateBytes(st, N)

3.2. Pseudo-random function

Algorithms 3 implements the {Forro, Xote}.PRF($\cdot$). This function receives a seed r, a nonce i, and the output length N as inputs. In this algorithm, {Forro, Xote}.Keysetup($\cdot$) sets r as a key while {Forro, Xote}.IVsetup($\cdot$) uses i as IV, initiating the state matrix st. Based on st and N, {Forro, Xote}.GenerateBytes($\cdot$) is called outputting a N-bytes string.

3.3. Key-derivation function

Algorithm 4 implements the {Forro, Xote}.KDF($\cdot$). This function receives 64-bytes of key material kr and operates on it to generate a shared secret of N bytes, which, in this

Algorithm 3 {Forro, Xote}.PRF(r, i, N)

Input:
Seed: $r \in \mathcal{B}^{32}$
Nonce: $i \in \mathcal{B}$
Output length: $N \in \mathcal{U}$
Output:
Byte string: $out \in \mathcal{B}^{*}$
Procedure:
State matrix: $st \in \mathcal{U}^{4\times4}$
IV: $iv \in \mathcal{B}^{32}$
$iv[0] = i$
{Forro, Xote}.Keysetup(st, r)
{Forro, Xote}.IVsetup(st, iv)
Return:
Byte string: out := {Forro, Xote}.GenerateBytes(st, N)

case, of the ML-KEM, is fixed to $N = 32$. Nonetheless, Forró and Xote can only use 32-bytes as a key and 16-bytes as a nonce. Consequently, two iterations are required to consume all bits of key material kr. Therefore, the first half of key material kr and iv are initialized in state matrix st_1, and the second half of kr and iv, incremented by one, are initialized in state matrix st_2. Both state matrices are *xored*, generating a state matrix st. Finally, a 32-bytes shared secret is achieved by performing {Forro, Xote}.GenerateBytes($\cdot$) with st as input.

4. Experimental results

This section analyzes the implementation of the modified versions of ML-KEM, based on Forró and Xote, and provides a comparison with the standard ML-KEM based on SHAKE. To provide comprehensive analyses, it details the execution time of the core functions—XOF, PRF, and KDF—using SHAKE, Forró, and Xote, separately from the ML-KEM. Moreover, it compares the execution time of composite functions—key pair generation, encapsulation, and decapsulation—using SHAKE, Forró, and Xote across all security levels. The performance results refer to several execution times conducted across different security levels. They are also presented as boxplots from a collection of 101 samples. Each sample represents the median of $10,001$ iterations of each function at each security level. The results were achieved in a Xeon E5-1650 v4 processor, using GCC compiler with $-O3$ optimization flag. It is important to mention that experiments ran in an Intel Core i5 10th Generation and Apple M2 Pro achieved similar results. The source code can be found in [Lagrota and Azevedo 2024]

This section is organized as follows: Subsection 4.1 pays attention to the results of the core functions; Subsection 4.2 details the results of the composite functions; and Subsection 4.3 discusses the execution time improvements of Forró and Xote compared to SHAKE.

Algorithm 4 {Forro, Xote}.KDF(kr, N)

Input:
Key material: $kr \in \mathcal{B}^{64}$
Output:
Shared secret: $ss \in \mathcal{B}^{32}$
Procedure:
State matrix: $st, st_1, st_2 \in \mathcal{U}^{4\times4}$
IV: $iv \in \mathcal{B}^{32}$
$iv := 0$
▷ Consuming first half of key material kr
{Forro, Xote}.Keysetup(st_1, kr)
{Forro, Xote}.IVsetup(st_1, iv)
▷ Consuming second half of key material kr
{Forro, Xote}.Keysetup($st_2, kr[32]$)
{Forro, Xote}.IVsetup($st_2, iv + 1$)
$st := st_1 \oplus st_2$
Return:
Shared secret: ss := {Forro, Xote}.GenerateBytes(st, N)

4.1. Core functions

Core functions, composed of XOF, PRF, and KDF, are functions in the ML-KEM that use symmetric cryptography to accomplish their tasks. Their timing analyses are presented in Subsections 4.1.1, 4.1.2, and 4.1.3, respectively.

4.1.1. XOF

Figure 2 shows the boxplot of the XOF-absorb execution time. The data shows that XOF-absorb runs faster when SHAKE is used compared to Forró and Xote for all security levels. This increased speed is attributed to the use of SHAKE-128. SHAKE-128 outperforms SHAKE-256 due to its higher rate and lower capacity, although it offers less security [Dworkin 2015]. Additionally, Forró runs faster than Xote, which is explained by Xote's architecture, which uses two state matrices instead of one, as Forró does. Consequently, when Forro.QR($\cdot$) and Xote.QR($\cdot$) are called in Algorithm 1, Xote processes two state matrices, doubling the time required compared to Forró.

On the other hand, in the XOF-squeeze, Xote is faster than SHAKE and Forró regardless of the security level, see Figure 3. This advantage is also due to the state matrices, but in this case, it benefits Xote. As shown in Algorithm 2, the only required function is {Forro, Xote}.GenerateBytes($\cdot$), which uses {Forro, Xote}.QR($\cdot$) to generate bytes. Consequently, the double-state matrix construction of Xote allows it to outperform both SHAKE and Forró in this context.

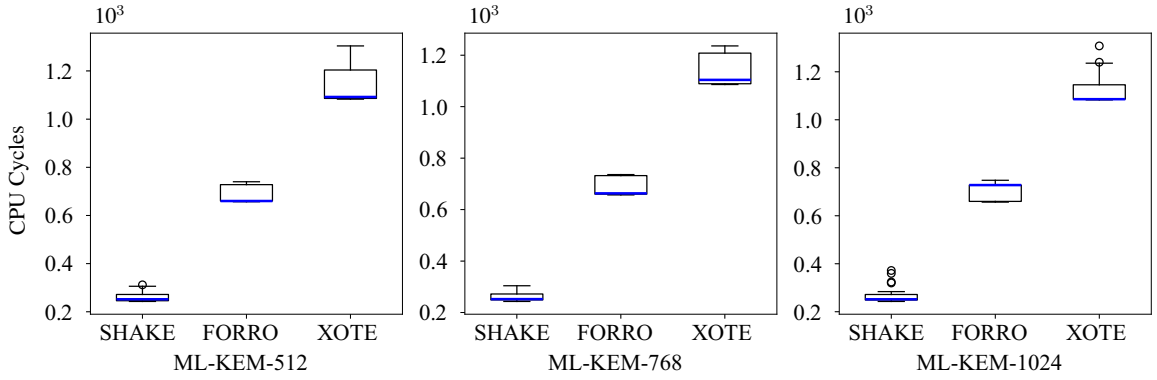


Figure 2. Boxplot of XOF-absorb execution time.

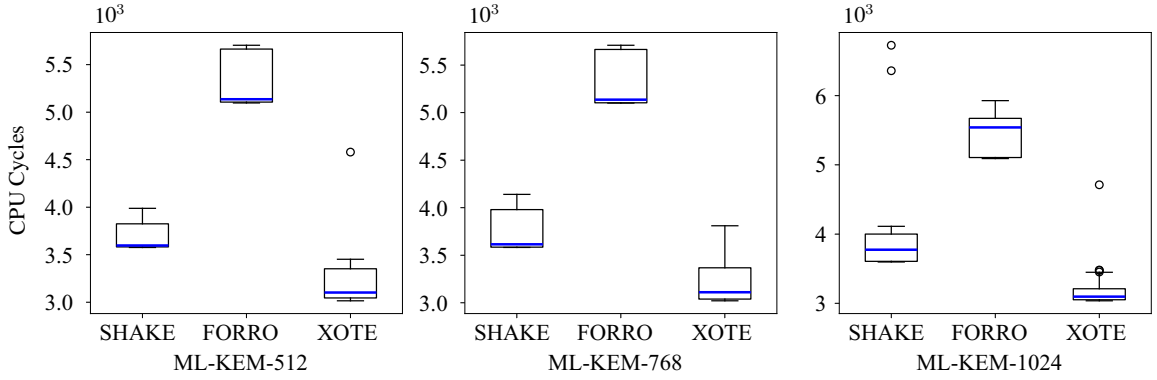


Figure 3. Boxplot of XOF-squeeze execution time.

4.1.2. PRF

The boxplot of the PRF execution time is shown in Figure 4 for all security levels. While SHAKE, Forró, and Xote exhibit similar performance, Xote slightly outperforms the other two, particularly at higher security levels. This is because {Forro, Xote}.PRF(·) relies on {Forro, Xote}.Encrypt(·), which in turn utilizes {Forro, Xote}.QR(·). The use of two-state matrices in Xote speeds up the process. As higher security levels require more pseudorandom data to be generated, the performance difference between SHAKE, Forró, and Xote becomes more pronounced with the increased data requirement.

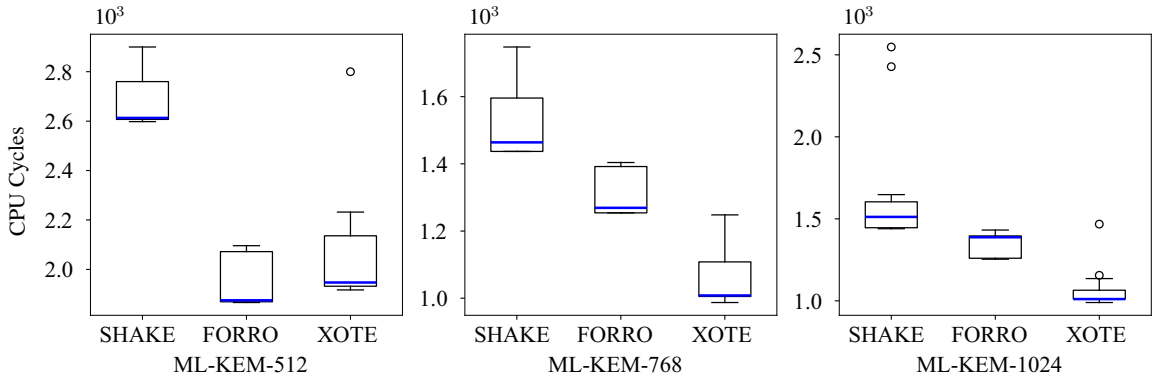


Figure 4. Boxplot of PRF execution time.

4.1.3. KDF

Figure 5 presents the boxplot of KDF execution time for all security levels. In this case, Forró outperforms Xote and SHAKE. The doubled state matrices work against Xote in this scenario. Although {Forro, Xote}.GenerateBytes(·) is used, the required data length is only 32-bytes. This amount can be generated in a single operation by one state matrix. Consequently, Xote processes more data than necessary, reducing its efficiency regardless of the security level.

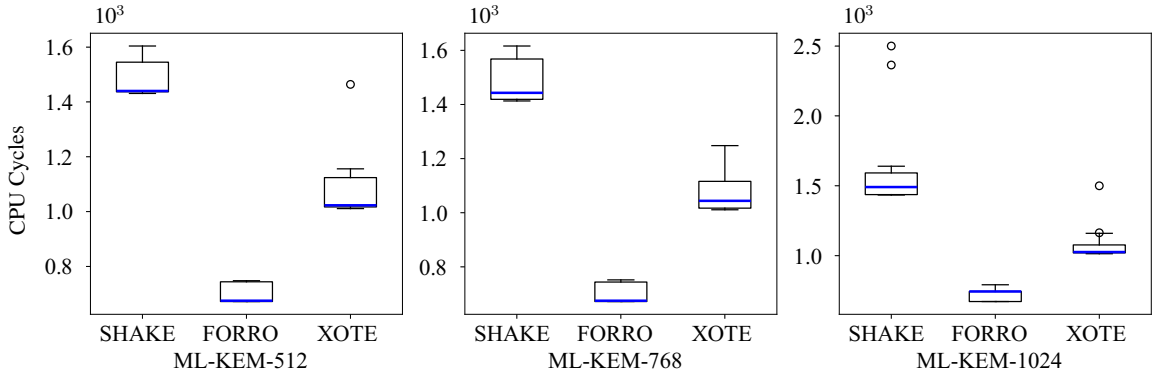


Figure 5. Boxplot of KDF execution time.

4.2. Composite functions

Composite functions are the primary interfaces of ML-KEM used by applications. These functions invoke various operations, including core functions. Table 1 details the frequency with which each core function is called in ML-KEM. It provides essential information for understanding the execution time of composite functions, as core functions directly influence their execution time. Next, Subsections 4.2.1, 4.2.2, and 4.2.3 present the execution time for key pair generation, encapsulation, and decapsulation, respectively.

Table 1. Number of times that XOF-absorb, XOF-squeeze, PRF, KDF are called in key pair generation, encapsulation, and decapsulation. $K \in \{2, 3, 4\}$ **refers to the security level of ML-KEM-**512**, -**768**, -**1024**, respectively.**

	XOF-absorb	XOF-squeeze	PRF	KDF
Key pair generation	K^2	K^2	$2K$	0
Encapsulation	K^2	K^2	$2K+1$	1
Decapsulation	K^2	K^2	$2K+1$	1

4.2.1. Key pair generation

Figure 6 presents the boxplots of the total execution time for key pair generation. It is important to note that key pair generation involves extensive use of XOF-absorb and XOF-squeeze functions (see Table 1) and is therefore significantly impacted by their execution times. XOF-squeeze has a greater influence among these functions due to its

longer total execution time. Consequently, the boxplot of key pair generation execution time closely resembles that of the XOF-squeeze function. Specifically, for all security levels, the ML-KEM instantiated with Xote demonstrates the best performance, followed by its versions based on SHAKE and Forró, in that order.

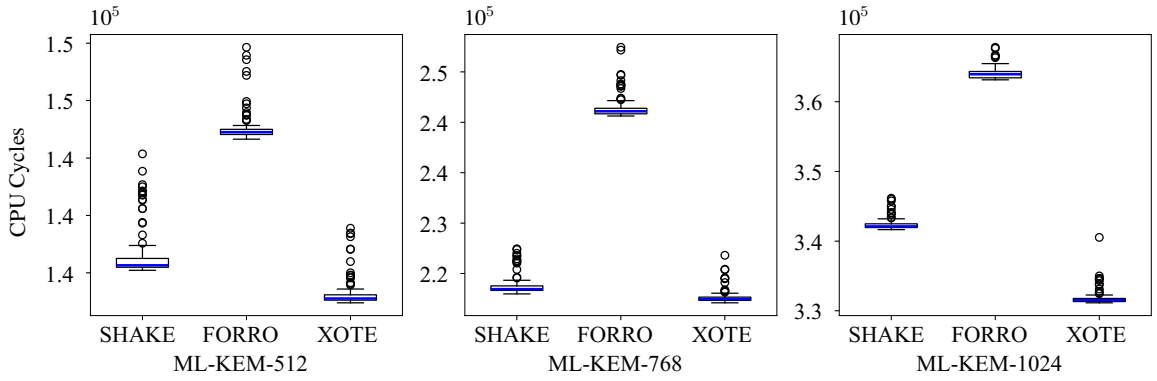


Figure 6. Boxplot of key pair generation execution time.

4.2.2. Encapsulation

The boxplot of encapsulation execution time is presented in Figure 7. The overall behavior of the results is similar to that of key pair generation for all security levels. Thus, the ML-KEM instantiated with Xote shows the best performance, followed by the versions based on SHAKE and Forró, respectively. These results are primarily due to the impact of XOF-squeeze, but also reflect the influence of an additional PRF. Encapsulation involves one more PRF and KDF compared to key pair generation. Consequently, the total execution time of encapsulation is greater, although the performance of the ML-KEM with different symmetric primitives follows the same order.

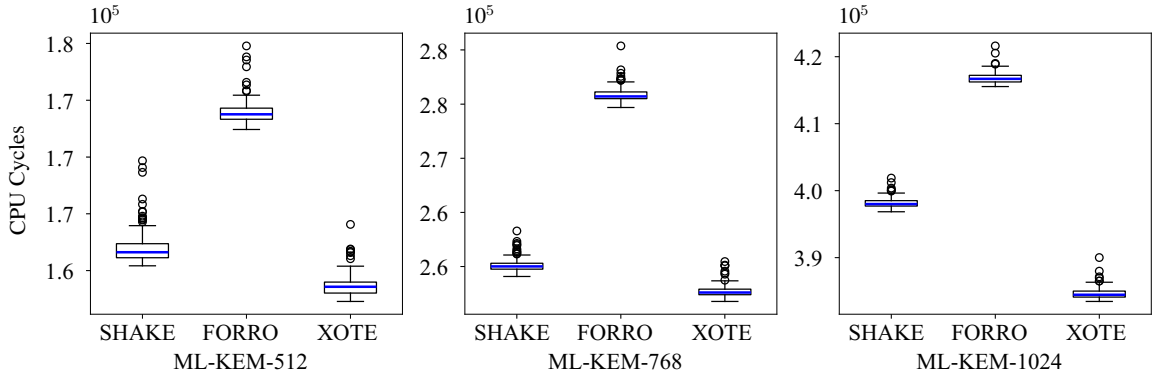


Figure 7. Boxplot of encapsulation execution time.

4.2.3. Decapsulation

Finally, the boxplot of decapsulation execution time is presented in Figure 8. Again, the ML-KEM instantiated with Xote is the fastest, with the SHAKE and Forró versions following in that order. These results follow the same pattern observed for key pair generation and encapsulation. Since encapsulation and decapsulation execute the same number

of symmetric primitive functions, similar performances are equally attributed to the impact of XOF-squeeze and PRF. However, note that the overall execution time for decapsulation is greater than for encapsulation. This difference is due to additional functions beyond symmetric primitives required only by decapsulation.

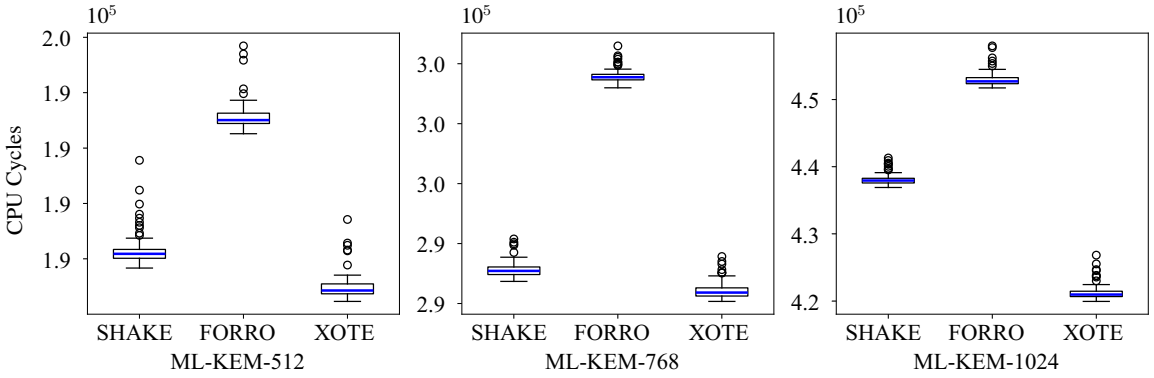


Figure 8. Boxplot of decapsulation execution time.

4.3. Execution time improvement

Table 2 presents the execution time improvement of key pair generation, encapsulation, and decapsulation when ML-KEM, Forró-based ML-KEM, and Xote-based ML-KEM are considered. The execution time improvement is given by

$$\gamma_{\alpha,\beta,K} = 1 - \frac{T_{\alpha,\beta,K}}{T_{\alpha,\text{SHAKE},K}}, \tag{1}$$

where $\alpha \in \{\text{key pair generation, encapsulation, decapsulation}\}$ and $\beta \in \{\text{Forró, Xote}\}$. $T_{\alpha,\text{SHAKE},K}$ and $T_{\alpha,\beta,K}$ refer to the execution times demanded by the ML-KEM and its modified versions, respectively.

Note that ML-KEM with Xote shows a small performance improvement over ML-KEM with SHAKE. This improvement is attributed to the efficient performance of Xote.XOF-squeeze(·) and Xote.PRF(·) (see Figures 3 and 4), which enable ML-KEM to be executed slightly faster with Xote than with SHAKE. Conversely, ML-KEM with Forró experiences a small performance decrease compared to ML-KEM with SHAKE. This is due to Forro.XOF-squeeze(·) significantly impacts the performance, making it slower than SHAKE in total execution time. The performance comparison remains consistent across all security levels (K).

5. Conclusion

This paper has explored replacing SHAKE with Forró and Xote as cryptographic primitives in ML-KEM. To achieve this, detailed modifications to the core functions of ML-KEM were made to fulfill Forró and Xote, and their integration with ML-KEM was discussed. Numerical results from experiments show that ML-KEM, Forró-based ML-KEM, and Xote-based ML-KEM exhibit distinct execution times for their core functions. Interestingly, the optimal performance for each core function is achieved by a different symmetric primitive. However, when evaluating the composite functions—key pair generation, encapsulation, and decapsulation—integral to a KEM, Xote-based ML-KEM

Table 2. Execution time improvement ($\gamma_{\alpha,\beta,K}$) of Forró- and Xote-based ML-KEM in percentage ($\%$).

	Algorithm	Forró	Xote
ML-KEM-512	Key pair generation	-4.13	1.03
	Encapsulation	-3.66	0.92
	Decapsulation	-3.22	0.88
ML-KEM-768	Key pair generation	-7.90	0.44
	Encapsulation	-6.04	0.93
	Decapsulation	-5.61	0.63
ML-KEM-1024	Key pair generation	-6.38	3.10
	Encapsulation	-4.70	3.40
	Decapsulation	-3.37	3.88

demonstrates slightly better performance than ML-KEM while maintaining equal or superior security across various security levels. Furthermore, the numerical results reveal that Forró-based ML-KEM is outperformed by both Xote-based ML-KEM and ML-KEM in terms of execution time, despite offering similar security. Overall, this work presented ML-KEM, a PQC algorithm, instantiated with Forró and Xote, two Brazilian cryptographic primitives, presenting better or similar performance. Future work should include a similar analysis incorporating AES with hardware acceleration and optimizing SHAKE, Forró, and Xote using Advanced Vector Extensions 2 (AVX2).

Acknowledgments

This research was supported by Centro de Pesquisa e Desenvolvimetno para a Segurança das Comunicações (CEPESC), Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) under Grant 001, Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) under grants $404068/2020-0$ and $314741/2020-8$, Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG) under grants APQ-$03609-17$, TEC-PPM $00787-18$, and APQ-04623-22, Instituto Nacional de Energia Elétrica (INERGE).

References

[Albrecht and Deo 2017] Albrecht, M. R. and Deo, A. (2017). Large modulus ring-LWE $\geq$ module-LWE. In *Proc. Int. Conf. on the Theory and Application of Cryptology and Information Security*, pages 267–296. Springer.

[ANSSI 2022] ANSSI (2022). Anssi views on the post-quantum cryptography transition. Technical report, ANSSI.

[Avanzi et al. 2021] Avanzi, R., Bos, J. W., Ducas, L., Eike Kiltz, T. L., Lyubashevsky, V., Schanck, J. M., Schwabe, P., Seiler, G., and Stehlé, D. (2021). CRYSTALS-Kyber

algorithm specifications and supporting documentation (version 3.0). Technical report, NIST, Gaithersburg, MD.

[Barbosa and Hülsing 2023] Barbosa, M. and Hülsing, A. (2023). The security of kyber's fo-transform. *IACR Cryptology ePrint Archive*, 2023(755).

[Bernstein 2008] Bernstein, D. J. (2008). The salsa20 family of stream ciphers. In *New stream cipher designs: the eSTREAM finalists*, pages 84–97. Springer.

[Bernstein et al. 2008] Bernstein, D. J. et al. (2008). Chacha, a variant of salsa20. In *Workshop record of SASC*, volume 8, pages 3–5.

[BSI 2020] BSI, C. M. (2020). Recommendations and key lengths. Technical report, BSI.

[Costa et al. 2022] Costa, V. L. R. D., Camponogara, Â., López, J., and Ribeiro, M. V. (2022). The feasibility of the crystals-kyber scheme for smart metering systems. *IEEE Access*, 10:131303–131317.

[Coutinho 2021] Coutinho, M. (2021). Forró and xote cipher. `https://github.com/murcoutinho/forro_cipher`.

[Coutinho et al. 2022] Coutinho, M., Passos, I., Grados Vásquez, J. C., de Mendonça, F. L. L., de Sousa, R. T., and Borges, F. (2022). Latin dances reloaded: Improved cryptanalysis against salsa and chacha, and the proposal of forró. In Agrawal, S. and Lin, D., editors, *Advances in Cryptology – ASIACRYPT 2022*, pages 256–286, Cham. Springer Nature Switzerland.

[Da Costa et al. 2022] Da Costa, V. L., Camponogara, Â., López, J., and Ribeiro, M. V. (2022). The feasibility of the crystals-kyber scheme for smart metering systems. *IEEE Access*, 10:131303–131317.

[Dworkin 2015] Dworkin, M. (2015). Sha-3 standard: Permutation-based hash and extendable-output functions.

[Jati et al. 2024] Jati, A., Gupta, N., Chattopadhyay, A., and Sanadhya, S. K. (2024). A configurable crystals-kyber hardware implementation with side-channel protection. *ACM Trans. Embed. Comput. Syst.*, 23(2).

[Lagrota and Azevedo 2024] Lagrota, V. and Azevedo, B. (2024). Ml-kem instantiated with forró and xote cipher. `https://github.com/beatrizufjf/kyber.git`.

[Langlois and Stehlé 2015] Langlois, A. and Stehlé, D. (2015). Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75(3):565–599.

[National Institute of Standards and Technology 2024] National Institute of Standards and Technology (2024). Module-lattice-based key encapsulation mechanism standard. Federal Information Processing Standards Publication (FIPS) NIST FIPS 203, Department of Commerce, Washington, D.C.

[NCSC 2020] NCSC (2020). Preparing for quantum-safe cryptography. Technical report, NCSC.

[Nguyen and Gaj 2021] Nguyen, D. T. and Gaj, K. (2021). Optimized software implementations of crystals-kyber, ntru, and saber using neon-based special instructions of armv8. In *Proceedings of the NIST 3rd PQC Standardization Conference (NIST PQC 2021)*.

[Pacheco et al. 2022] Pacheco, R., Braga, D., Passos, I., Araújo, T., Lagrota, V., and Coutinho, M. (2022). libharpia: a new cryptographic library for brazilian elections. In *Anais do XXII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 250–263. SBC.

[Shor 1994] Shor, P. W. (1994). Algorithms for quantum computation: Discrete logarithms and factoring. In *Proc. 35th Annual Symposium on Foundations of Computer Science*, pages 124–134.

[Wan et al. 2022] Wan, L., Zheng, F., Fan, G., Wei, R., Gao, L., Wang, Y., Lin, J., and Dong, J. (2022). A novel high-performance implementation of crystals-kyber with ai accelerator. In Atluri, V., Di Pietro, R., Jensen, C. D., and Meng, W., editors, *Computer Security – ESORICS 2022*, pages 514–534, Cham. Springer Nature Switzerland.

[Xing and Li 2021] Xing, Y. and Li, S. (2021). A compact hardware implementation of cca-secure key exchange mechanism crystals-kyber on fpga. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 328–356.

PARDED
Uma ferramenta de detecção passiva de malwares com foco em rootkits que utilizam técnicas de ofuscação de tráfego

Maickel J. Trinks[1], **Mateus Terra**[3], **João Gondim**[2]

[1]Centro de Pesquisa e Desenvolvimento para a Segurança das Comunicações (CEPESC), Brasília-DF, Brasil

[2]Departamento de Engenharia Elétrica - Universidade de Brasília (UnB), Brasília-DF, Brasil

[3]Programa de Pós-Graduação Profissional em Engenharia Elétrica - PPEE, Departamento de Engenharia Elétrica - Universidade de Brasília (UnB), Brasília-DF, Brasil

{maickeljosue@yahoo.com.br, gondim@unb.br, mateus.b.s.terra@gmail.com}

Abstract. *PARDED is a passive malware detection tool, focusing on rootkits that use traffic obfuscation techniques. The system detects malicious behavior through a multi-agent system, installed in the analyzed terminals and in the network infrastructure, verifying suspicious data flows and enriching information with local and remote databases, in addition to having an intuitive visualization interface and generation of traffic blocking rules and cyber threat intelligence, enabling integration with existing conventional defense systems, without affecting the connection performance of the terminals.*

Resumo. *PARDED é uma ferramenta de detecção passiva de malwares, com foco em rootkits que utilizam técnicas de ofuscação de tráfego. O sistema detecta comportamento malicioso através de um sistema multiagente, instalado nos terminais analisados e na infraestrutura de rede, verificando fluxos de dados suspeitos, enriquecendo informações com bases de dados locais e remotas, além de possuir interface de visualização intuitiva e geração de regas de bloqueio de tráfego e inteligência de ameaças cibernéticas, permitindo integração com sistemas de defesa convencionais existentes, sem afetar o desempenho de conexão dos terminais.*

1. Introdução

Passive Rootkit Detector With Enriched Data (PARDED) é uma arquitetura multiagente, composta por elementos que trabalham em pontos distintos de uma infraestrutura de rede de comunicação, onde cada componente integrante é autônomo, como em [Julian and Botti 2019]. Sua função é detectar técnicas de ofuscação de tráfego em terminal infectado por software malicioso. Tais *malwares* impedem que análises locais, tanto pela verificação da tabela de conexões quanto por captura de tráfego local, detectem a transmissão de dados maliciosos.

Foram encontradas apenas duas publicações que utilizam a abordagem de detecção baseada em rede, nas quais foram apresentadas as arquiteturas MADEX

[Marques et al. 2021] e NERD [Terra and Gondim 2021]. O PARDED visa aprimorar a performance dessas arquiteturas e melhorar a efetividade da detecção ao eliminar gargalos na forma como o agente auditor atua para detecção e o bloqueio da comunicação maliciosa, de forma passiva [Trinks et al. 2023], a fim de evitar degradação na taxa de transmissão dos equipamentos de rede, além de utilizar dados enriquecidos para decisões de bloqueio de tráfego e prover integração com outros sistemas de defesa.

O PARDED possui como principal característica o aperfeiçoamento das técnicas de bloqueio em infraestruturas de redes de comunicação sem degradar a taxa de transmissão de dados nos equipamentos de rede, além de prover uma interface para integração com outros sistemas de defesa previamente existentes, como *firewalls* e sistemas de detecção de intrusão (IDS). A ferramenta permite ainda a integração com sistemas de inteligência de ameaças, com viés cibernético (CTI), conforme definições apresentadas em [Tounsi and Rais 2018].

Outra característica importante adicionada é a possibilidade de enriquecimento das informações acerca dos destinos maliciosos utilizados pelos *rootkits*, notadamente para sistemas de comando e controle (C2), auxiliando o analista de segurança com dados importantes para análise comportamental, outros comprometimentos registrados para o IP detectado, informações existentes acerca do domínio malicioso, entre outros. O enriquecimento é realizado através de bases de informação locais ou externas, como nomes de domínio consultados pelo *malware*, geolocalização, lista de IPs comprometidos ou análises de empresas de segurança cibernética.

O conceito da arquitetura PARDED foi apresentado no *IV Congreso de Ciência de la Computacion, Eletronica e Industrial* - CSEI 2022 [Trinks et al. 2023], com o título *Multi-agent Architecture for Passive Rootkit Detection with Data Enrichment*, onde foram descritas as melhorias arquiteturais que o PARDED introduz ao NERD [Terra and Gondim 2021] e ao MADEX [Marques et al. 2021]. Neste artigo apresenta-se a implementação do PARDED, agregando à detecção um arcabouço para enriquecimento dos alertas e integração com ferramentas de segurança, levando a um *inicial report*. Assim, elevamos o grau de maturidade de detecção [Bromander et al. 2016] para uma caracterização que pode correlacionar alertas de TTP (Táticas, Técnicas e Procedimentos) e até possíveis artefatos e agentes maliciosos.

O artigo está organizado da seguinte forma: a seção 2 apresenta a arquitetura e as funcionalidades da ferramenta; a seção 3 detalha o desempenho em diversos cenários de teste; a seção 4 descreve a demonstração realizada com a ferramenta; a seção 5 explica com mais detalhes onde encontrar o código do sistema e instalação a solução; por fim, a seção 6 apresenta as conclusões.

2. Arquitetura e Funcionalidades

A arquitetura PARDED foi desenvolvida nas linguagens Nim, C e Python 3, aproveitando as principais características das diferentes linguagens de programação em cada um dos elementos e subsistemas, como velocidade de processamento, facilidade de uso de APIs existentes e possibilidade de criação de interfaces de visualização interativas. O sistema é composto por quatro elementos principais, de forma integrada, conforme Figura 1. Cada um dos elementos é detalhado nas subseções abaixo.

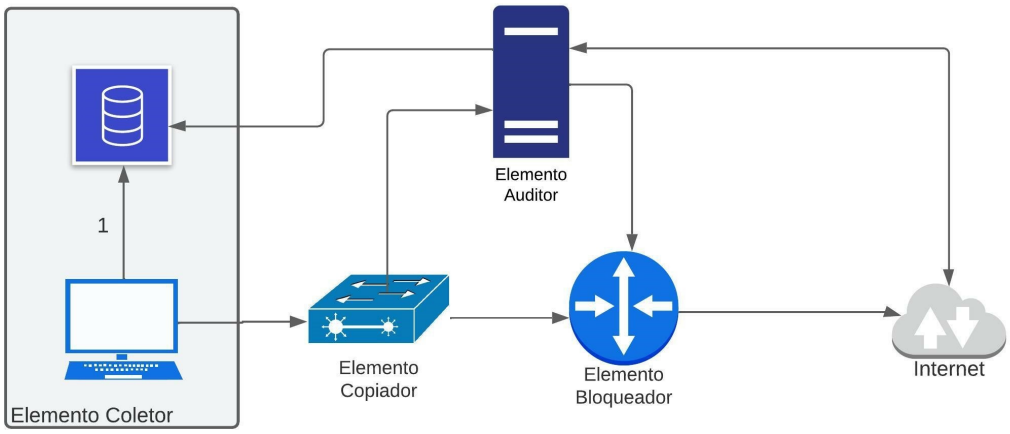


Figura 1. Arquitetura PARDED proposta (Figura do Autor)

2.1. Elemento Coletor (ECol)

O Elemento Coletor é inserido no terminal a ser analisado, verificando o tráfego através da captura de tráfego pelo próprio terminal, com auxílio da biblioteca `libpcap`. Em virtude dos bons resultados apresentados, optou-se por utilizar a abordagem prevista na arquitetura NERD [Terra and Gondim 2021]. Os pacotes de dados, ao serem transmitidos pelo terminal analisado, são detectados pelo ECol e armazenados em uma base intitulada Base de Fluxos. Os pacotes gerados por *rootkits* que ofuscam o tráfego de rede não são identificados e, portanto, não serão adicionados à Base de Fluxos. Dessa forma, é possível a um elemento exterior diferenciar tráfegos legítimos de ofuscados.

2.2. Elemento Auditor (EAud)

O elemento auditor possui a função de comparar o tráfego capturado na rede com o percebido pelo Coletor. Assim, fluxos que foram ofuscados e não foram detectados pelo ECol, ao trafegarem na rede, serão detectados pelo EAud e, ao ser comparado com a tabela de fluxos do ECol, classificados como suspeitos. o EAud não executa internamente a função de bloqueio, permitindo que trabalhe *out-of-band*. Dessa forma, análises mais complexas podem ser realizadas antes do bloqueio e sem impacto de desempenho na comunicação do terminal, ou seja, sem afetar performance de transmissão ou tráfego em tempo real.

O EAud foi separado em sistemas (módulos), conforme Figura 2, facilitando alterações nos métodos de captura e análise, inserção de novas bases de enriquecimento e execução das etapas de processamento de forma assíncrona.

O Sistema de Análise Inicial foi desenvolvido em linguagem C, em virtude de ser o módulo mais crítico e que precisa ser performático. Ele utiliza uma lista de bibliotecas disponíveis para linguagem C, em especial a utilização da biblioteca `etcdlib.h` para comunicação com o coletor, `libpq-fe.h` para comunicação com o banco de dados relacional, padrão SQL (PostgreSQL), e `pcap.h` para criação de um *handle* de captura de pacotes. A cada pacote, o sistema verifica se o fluxo é conhecido, através da Base Temporária (em memória RAM). Caso não seja conhecido, é feita consulta ao ECol. Se for considerado legítimo, é adicionado na Base Temporária; caso contrário, na Base de Conexões (fluxos suspeitos). Pacotes DNS Response também são tratados e a relação de IPs e domínios é adicionada na Base DNS.

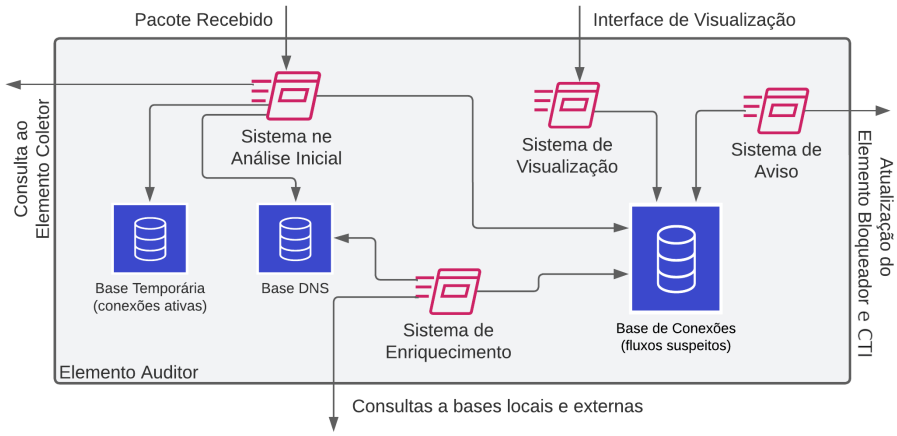


Figura 2. Elemento Auditor da estrutura PARDED (Figura do Autor)

O Sistema de Enriquecimento verifica, através da Base de Conexões e de forma assíncrona, quais dados ainda não foram enriquecidos e consulta as bases (locais e externas) configuradas. O Sistema de Enriquecimento atualiza então a Base de Conexões com as novas informações obtidas. Foi implementado em Python 3.

O Sistema de Aviso verifica, através da Base de Conexões e também de forma assíncrona, quais destinos atingiram os limiares configurados para que um fluxo suspeito seja considerado malicioso. Quando um limiar é atingido, o sistema cria arquivos de aviso ou bloqueio e repassa a regra ao Elemento Bloqueante. Essas regras devem ser customizadas de acordo com a API disponibilizada pelo Elemento Bloqueante, seja ele um firewall ou IPS. Na implementação apresentada, os arquivos de bloqueio estão no formato *Snort*. São criados, ainda, arquivos de CTI no formato STIX 2.1, permitindo integração com sistemas de inteligência de ameaças. Também foi implementado em Python 3.

O Sistema de Visualização permite que o analista de segurança verifique os fluxos assinalados como suspeitos e suas características, além de um panorama de funcionamento da própria ferramenta, facilitando a utilização em ambientes de monitoramento e integração com demais sistemas de segurança. Implementado em linguagem Python 3, com auxílio do *framework* de visualização de dados *Dash*, facilitando a criação de *dashboards* interativos e atualizações em tempo real.

2.3. Elemento Copiador

A função do Elemento Copiador é permitir o funcionamento passivo do Elemento Auditor, espelhando o tráfego que passa por ele. Assim, o tráfego original é encaminhado para o próximo elemento de rede e sua cópia é enviada para análise do EAud, evitando atrasos na transmissão de dados ao destino.

A arquitetura PARDED não define como deve ser implementado o Elemento Copiador, pois é possível a utilização de qualquer equipamento existente na infraestrutura que realize cópias dos pacotes trafegados, como comutadores, roteadores, hubs ou firewalls, facilitando a integração em redes complexas.

2.4. Elemento Bloqueante (EBlk)

O Elemento Bloqueante (EBlk) é um equipamento *inline* que verifica se o pacote de rede recebido deve ou não ser bloqueado, baseado no aviso recebido pelo EAud. Esse elemento pode ser qualquer dispositivo que permita receber atualizações de regras de alerta ou bloqueio de forma remota, como roteadores e firewalls.

3. Desempenho

O desempenho da arquitetura foi testado através da geração de tráfego e utilização de um *rootkit* em ambiente controlado. Para tal, foram utilizados os softwares `wget` e `iperf` para criar tráfego legítimo e o Rootkit Linux Nuk3Gh0st [Schällibaum 2019] para o tráfego malicioso (entretanto, qualquer rootkit que utiliza técnicas de ofuscação de tráfego pode ser utilizado). Tanto o Elemento Coletor quanto o Auditor foram executados em máquinas virtuais com o sistema operacional Linux Debian 11 (*bullseye*) (x86_64, *release* 5.10.0-16, *version* 5.10.127-1).

O primeiro teste foi executado sem atuação de ofuscação de tráfego, ou seja, apenas com fluxos de dados legítimos. Em seguida, foram realizados testes com transmissão apenas de fluxos maliciosos. Por fim, um terceiro cenário foi gerado para testes de detecção de fluxos legítimos e suspeitos trafegando concomitantemente.

3.1. Fluxos Legítimos

Nesse cenário, o *rootkit* não estava ativo e tráfego legítimo foi gerado a partir do ECol, com espelhamento de tráfego ao EAud. O resultado esperado era a detecção de todos os fluxos como legítimos, com inclusão dos fluxos conhecidos na Base Temporária (EAud) e das respostas de consultas DNS inseridas na Base DNS (EAud). O teste, contendo de 50 a 60 mil pacotes, foi executado 30 vezes. O desempenho obtido está demonstrado na Tabela 1, com a média de pacotes por fluxo de cada execução e o tempo de processamento por pacote (com e sem carga de 50Mbps).

Tabela 1. Tempo de resposta do sistema (sem atuação do *rootkit*).

Tipo de Processamento	Característica do Fluxo	Média de Pacotes	Tempo de processamento (ms)	Tempo de proces. com carga de 50Mbps(ms)
Sem consulta ao Coletor	Base Temporária	55.393	0,00110	0,00100
	Resposta DNS	2030	0,00407	0,00734
Com consulta ao Coletor	Fluxos Legítimos	983	516,78	551,05

Todos os pacotes recebidos no EAud foram percebidos corretamente como legítimos. Os pacotes que não estavam presentes na Base Temporária do EAud (o que requer consulta ao ECol) foram, em média, processados em 0,5 segundos. Os demais pacotes do fluxo (já inseridos na Base Temporária), 95% do total gerado, foram processados em menos de 0,0012 milissegundos cada pelo Sistema de Análise Inicial. Os pacotes "DNS response", tratados localmente para detecção dos possíveis domínios utilizados pelos *rootkits*, foram processados em aproximadamente 0,0041 milissegundos.

Tabela 2. Tempo de resposta do sistema pra fluxos exclusivamente suspeitos.

Tipo de Processamento	Característica do Fluxo	Média de Pacotes	Taxa (pacotes por segundo)	Tempo de processamento (ms)
Com consulta ao Coletor	Fluxos Suspeitos	100	1 pps	1.742,77
		100	10 pps	1.674,18
		500	10 pps	1.838,37

3.2. Fluxos Exclusivamente Maliciosos

Nesse cenário, o desempenho foi medido através da geração de pacotes maliciosos (não detectados pelo ECol). Foram utilizados apenas pacotes TCP SYN com porta de origem diferente para cada pacote, garantindo que cada um dos pacotes de dados transmitidos fosse detectado pelo EAud como um novo fluxo. Essa abordagem permitiu verificar o desempenho do sistema no pior cenário, onde cada pacote gera uma consulta ao ECol e gera também a necessidade de atualização da Base de Conexões. Os resultados estão presentes na Tabela 2. O teste foi executado 30 vezes para cada um dos 3 cenários apresentados (100 pacotes a 1pps, 100 pacotes a 10pps e 500 pacotes a 10 pps).

Todos os pacotes recebidos e tratados no EAud foram percebidos corretamente como suspeitos, ou seja, não houve falsos-positivos. O tempo de resposta foi, em média, de 1,71 segundos. A elevação do tempo de processamento deve-se ao fato do tratamento e atualização da base de conexões suspeitas, comprovando a diferença entre utilização de API do banco de dados relacional PostgreSQL e consultas diretas em memória volátil.

3.3. Fluxos Legítimos e Maliciosos

Para detecção de fluxos suspeitos em uma situação próxima a uma rede real, foram transmitidos ao mesmo tempo pacotes legítimos e maliciosos. Nesse cenário, o *rootkit* estava ativo e foi realizada transmissão de tráfego ofuscado entre os fluxos legítimos, correspondente a aproximadamente 2% do tráfego sem carga (100 fluxos de 10 pacotes cada), com intuito de verificar se os pacotes maliciosos seriam corretamente detectados e os dados inseridos na Base de Conexões. O teste, contendo de 50 a 60 mil pacotes, foi repetido 30 vezes. Os resultados estão demonstrados na tabela 3, com a média de pacotes por fluxo de cada execução e o tempo de processamento por pacote (com e sem carga).

Tabela 3. Tempo de resposta do sistema (com atuação do *rootkit*).

Tipo de Processamento	Característica do Fluxo	Média de pacotes	Tempo de processamento (ms)	Tempo de proces. com carga de 50Mbps(ms)
Sem consulta ao Coletor	Base Temporária	54.058	0,00114	0,00100
	Resposta DNS	1.329	0,0370	0,00730
Com consulta ao Coletor	Fluxos Legítimos	684	536,15	561,26
	Fluxos Suspeitos	100	1.642,64	1.709,62

Não houve alteração significativa no tempo de processamento de pacotes suspeitos, de *DNS Response* ou de pacotes presentes na Base Temporária. Todos os pacotes foram detectados corretamente, sem falsos-positivos ou falsos-negativos.

Tabela 4. Ações do Sistema de Enriquecimento.

Tipo de Processamento	Descrição da Base de Dados	Número de Consultas	Resposta (ms)
Consulta à base local	Nó pertencente a rede TOR	100	42,18
Consulta à base remota	Plataforma VirusTotal	30	1.241,14

3.4. Enriquecimento

Os testes de enriquecimento foram realizados com divisão em dois tipos de consulta: local, realizado com auxílio de base de dados de nós da rede TOR [Dingledine et al. 2004], obtida através do website *dan.me.uk* [Dan 2022]; e remoto, realizados em tempo real através de consultas na API da plataforma Virustotal [Chronicle Security 2023]. A ferramenta possui ainda geolocalização dos endereços de destino, obtida através da base GeoLite2-City, disponibilizada pelo empresa Maxmind Inc. [Maxmind 2023].

As consultas do Sistema de Enriquecimento feitas utilizando a consulta local (base de dados de nós TOR) foram processadas em aproximadamente 42 milissegundos cada. Em média, foram necessários cerca de 1,2 segundos para consultas do Sistema de Enriquecimento na base remota. Os resultados estão apresentados na Tabela 4.

3.5. Detecção de Limiares

Para execução dos testes de desempenho do Sistema de Aviso, foram definidos dois cenários possíveis: o primeiro, quando a consulta não encontra fluxos que atingiram os limiares de detecção; o segundo, quando um limiar de detecção é atingido e ações posteriores são executadas (criação de regra de bloqueio no formato Snort e CTI no formato STIX). Os resultados estão demonstrados na Tabela 5.

Tabela 5. Ações do Sistema de Aviso.

Tipo de Processamento	Número de Consultas	Resposta (ms)
Sem bloqueio (limiares não atingidos)	50	9,31
Com bloqueio (limiares atingidos)	50	89,60

4. Demonstração

A demonstração a ser apresentada, compilada nos vídeos Auditor (https://youtu.be/oCjbl4vpyO8) e Coletor (https://youtu.be/MBCC6ZzzPZk), descreve as principais funcionalidades da ferramenta, executando a geração de tráfego legítimo e malicioso, permitindo a visualização das técnicas de análise do tráfego, de enriquecimento dos dados e de atingimento de limiares de detecção (com geração dos arquivos de inteligência de ameaças cibernéticas e regras de bloqueio), além da interface visualização de fluxos suspeitos.

5. Código e manuais

O código fonte encontra-se no endereço https://github.com/maickelj/parded. O usuário e a senha para acesso, caso necessário, é “parded2023sbseg”. O arquivo README contém

as instruções necessárias para execução do código, requisitos de instalação e compilação. Os vídeos “auditor.mp4” e ”coletor.mp4”contêm, assim como o conteúdo dos links apresentados na seção 4, a demonstração do funcionamento da solução.

6. Conclusão

Esse trabalho apresentou uma arquitetura capaz de detectar *rootkits* que utilizam técnicas de ofuscação de comunicação no terminal infectado, de forma passiva, escalável, generalizável e adaptável, sem impactos significativos no desempenho da infraestrutura de rede, com enriquecimento de dados de múltiplas fontes e incorporação das informações oriundas de múltiplos terminais, permitindo integração com outros sistemas de defesa previamente existentes, como IDS e firewalls.

Portanto, o PARDED permite novos estudos de técnicas de detecção e bloqueio de *malwares* baseados em rede, enriquecimento de informações maliciosas e geração de *insights* para analistas de segurança cibernética.

Referências

Bromander, S., Jøsang, A., and Eian, M. (2016). Semantic cyberthreat modelling. *STIDS*, pages 74–78.

Chronicle Security (2023). Virustotal. http://www.virustotal.com. Acessado em 15 jul 2023.

Dan (2022). Tor Node List. https://www.dan.me.uk. Acessado em 12 jul 2023.

Dingledine, R., Mathewson, N., and Syverson, P. (2004). Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC.

Julian, V. and Botti, V. (2019). Multi-Agent Systems. *Applied Sciences (Switzerland)*, 9(7).

Marques, R. S., Epiphaniou, G., Al-Khateeb, H., Maple, C., Hammoudeh, M., De Castro, P. A. L., Dehghantanha, A., and Choo, K. K. R. (2021). A Flow-based Multi-Agent Data Exfiltration Detection Architecture for Ultra-low Latency Networks. *ACM Transactions on Internet Technology*, 21(4).

Maxmind (2023). GeoLite2 Free Geolocation Data. https://dev.maxmind.com/geoip/geolite2-free-geolocation-data?lang=en. Acessado em 24 Fev 2023.

Schällibaum, J. A. (2019). Nuk3 gh0st. https://github.com/juanschallibaum/Nuk3Gh0st. Acessado em 10 jul 2023.

Terra, M. B. and Gondim, J. J. (2021). NERD: A Network Exfiltration Rootkit Detector based on a Multi-agent Artificial Immune System. *2021 Workshop on Communication Networks and Power Systems, WCNPS 2021*.

Tounsi, W. and Rais, H. (2018). A survey on technical threat intelligence in the age of sophisticated cyber attacks. *Computers & Security*, 72:212–233.

Trinks, M., Gondim, J., and Albuquerque, R. (2023). Multi-agent architecture for passive rootkit detection with data enrichment. In *CSEI*, pages 29–41, Cham. Springer Nature Switzerland.

Artigos publicados em 2025

A Quantum-Resistant Advanced Metering Infrastructure

Vinícius Lagrota[1], Mateus de L. Filomeno[2], Leonardo de M. B. A. Dib[3], Julio López[4], Moisés V. Ribeiro[2]

[1] Research and Development Center for Communication Security (CEPESC)
Brasília, DF – Brazil.

[2]Electric Engineering Department
Federal University of Juiz de Fora (UFJF) – Juiz de Fora, MG – Brazil.

[3]WEG Electric Corp. – Jaraguá do Sul, SC – Brazil.

[4]Institute of Computing
Campinas State University (UNICAMP) – Campinas, SP – Brazil.

vinicius.lagrota@gmail.com, delimafilomeno@gmail.com,
ldib@weg.net, jlopez@ic.unicamp.br, moises.ribeiro@ufjf.br

Abstract. *This paper focuses on quantum-resistant cryptographic schemes to enhance the security of smart metering against threats from both quantum and classical computers. In this sense, we consider FrodoKEM and CRYSTALS-Kyber quantum-resistant as asymmetric cryptographic schemes and AES-GCM as a symmetric one. Moreover, we detail the design and implementation of a dedicated cryptographic module customized to ensure quantum-resistant advanced metering infrastructure (QR-AMI). Analysis based on experimental implementation shows the viability of a QR-AMI anchored on dedicated cryptographic modules implemented on a microcontroller (low-power processor) or a system-on-a-chip device.*

1. Introduction

Smart Grids (SGs) have materialized as an urgent solution to overcome the challenges in the electricity sector. These challenges encompass managing a plethora of loads, integrating renewable energy sources, and ensuring reliability, efficiency, sustainability, and flexibility, among others [Ghasempour and Lou 2017]. Yet, this influential paradigm only gains traction when stakeholders communicate effectively, thereby expediting the SG proliferation, facilitating a digital revolution in electric power systems globally [Ghasempour 2019], and unfortunately raising security and privacy concerns [Naderi and Asrari 2023].

In this scenario, Smart Meters (SMs) emerged as catalysts for large-scale SG deployments, having indisputable benefits. However, SMs generate and handle sensitive data of consumers, prosumers, and electric utilities. Consequently, a heightened awareness of security and privacy vulnerabilities in Advanced Metering Infrastructures (AMIs) has steered research and industrial endeavors toward protections against security infringements and unauthorized SM data access within AMI [Li et al. 2017]. Predominantly, these strategies rely on cryptography to safeguard against threats from classical computers in data networks, as evidenced in [Wang and Lu 2013]. However, the impending rise of

stable quantum computers threatens the current state of security, especially asymmetric cryptography, which is more vulnerable to quantum computational capabilities.

When it comes to asymmetric cryptography, its security is typically based on the complexity of integer factorization or the calculation of discrete logarithms, as in Rivest-Shamir-Adleman (RSA) and Elliptic Curve Cryptography (ECC) schemes [Wang and Lu 2013]. Unfortunately, these cryptographic schemes will be made insecure when cryptographic relevant quantum computers execute Shor's algorithm, a polynomial-time quantum algorithm for integer factorization and discrete logarithm operations [Shor 1994].

Integrating quantum-resistant asymmetric and symmetric cryptographic schemes is imperative to thoroughly neutralize the threats posed by quantum computers. In this context, studies centered on asymmetric post-quantum cryptography (PQC) adapted for SG and SM are paramount. In this domain, Cheng *et al.* [Cheng et al. 2019] unveiled a mutual authentication scheme between SMs and gateways, based on PQC. Similarly, Borges *et al.* [Borges et al. 2020] delivered a comparative analysis of key agreement protocols between PQC primitives and classical ones. Extending this, Ahn *et al.* [Ahn et al. 2022] assessed quantum-resistant key distribution frameworks, utilizing both Quantum Key Distribution (QKD) and PQC techniques, within Distributed Energy Resources (DER). More recently, the authors of [da Costa et al. 2022b] and [da Costa et al. 2022a] delved into the execution of the FrodoKEM and CRYSTALS-Kyber schemes for SMs. They established that a hardware/software co-design, built on System-on-a-Chip (SoC) devices, could feasibly incorporate PQC schemes within hardware-limited resources.

Despite recent efforts to address numerous concerns related to PQC for AMI, advancing the implementation of AMI based on quantum-resistant asymmetric and symmetric cryptographic schemes is imperative. These considerations are timely and critical, highlighting the merits of incorporating quantum security to counter threats from quantum and classical computers in AMI. Notably, as large-scale AMI deployments necessitate cost-effective solutions, discerning the suitable technologies that support PQC schemes within AMI is crucial.

To protect key agreements and sensitive data transactions between consumers/prosumers and electric utilities against potential quantum and classical computers, this paper delves into Quantum-Resistant AMI (QR-AMI) and the implementation nuances of the Quantum-Resistant Dedicated Cryptographic Module (QR-DCM), an integral and essential component of QR-AMI. It is important to mention that, regarding asymmetric cryptography, we focus exclusively on implementing PQC within our QR-AMI to address the emerging threat posed by quantum computing capabilities. While the most robust approach would involve hybrid cryptography [Barker et al. 2024] — combining PQC with classical cryptographic techniques like ECC to leverage their complementary strengths — we have chosen to limit our scope to PQC alone. This decision allows us to concentrate on the specific impact of quantum-resistant algorithms without the added complexity of hybrid solutions. Notably, ECC is significantly less memory-intensive than PQC. Thus, implementing a hybrid scheme would not substantially alter our hardware performance outcomes substantially, underscoring the feasibility of extending this work with hybrid techniques in future iterations. The main contributions of this paper encompass:

- A discussion on a QR-AMI harnessing either FrodoKEM or CRYSTALS-Kyber

for asymmetric cryptography and the Advanced Encryption Standard (AES)-256-Galois Counter Mode (GCM) for symmetric cryptography. This includes detailing the QR-DCM to enable quantum-resistant key agreements and the quantum-resistant encryption, decryption, and authentication of sensitive data transmitted through data networks.
- An implementation of the QR-DCM based on a low-power processor and high-power processor (i.e., software-driven), and SoC devices for hardware/software co-design.
- A comparative analysis that contrasts low-power and high-power processor-based implementations with the one on SoC devices, focusing on timing, hardware resource, and energy consumption.

The remainder of this paper is structured as follows: Section 2 formulates the problem under investigation. Section 3 details the QR-AMI based on QR-DCMs. Section 4 details cryptographic schemes chosen for a QR-DCM. Section 5 explores the implementation aspects of the QR-AMI, with a primary focus on QR-DCM. Performance comparisons are presented in Section 6. Section 7 states concluding remarks.

2. Problem Formulation

Smart metering enables monitoring and controlling energy usage of consumers/prosumers with more granularity and encompasses several functionalities [Bui et al. 2012]. These functionalities are met with data networks for exchanging sensitive information from consumers, prosumers, and utilities. In this context, AMI arises. It supports the acquisition of data by SMs, orchestrates communication among key stakeholders (namely consumers/prosumers and utility providers), and facilitates data storage and analytical processing within the Meter Data Management System (MDMS) [Mohassel et al. 2014].

Given the multi-user nature of AMI, the potential security attack surface is expansive, especially when data traverses potentially non-secure networks. A case in point is that even partial access to a subset of SMs data might provide an eavesdropper with insights that expose vulnerabilities in the critical electric power infrastructure. Furthermore, unauthorized data access at the individual SM level has the potential to reveal details about a consumer's behavioral patterns, including their daily routines and socioeconomic status. Recognizing these vulnerabilities, enhanced security protocols have been incorporated into AMI frameworks. These security measures bolster confidentiality and privacy and ensure the robustness of operations even when an attacker might leverage a classical supercomputer.

Figure 1 illustrates a novel and powerful threat scenario. At its core, a vulnerability in the data network becomes evident when an attacker, namely an eavesdropper, harnesses the power of a quantum computer. This capability allows unauthorized access to sensitive information exchanged between consumers or prosumers (represented by SMs) and electric utilities (denoted as MDMS). To effectively address this challenge, it is required to integrate specific cryptographic primitives into AMI. Specifically, quantum-resistant asymmetric and symmetric cryptographic schemes are vital to counter the potent threats posed by quantum computers.

No standard or regulation currently contemplates a PQC scheme in AMI. Consequently, a critical area of investigation centers on the viability of embedding quantum-

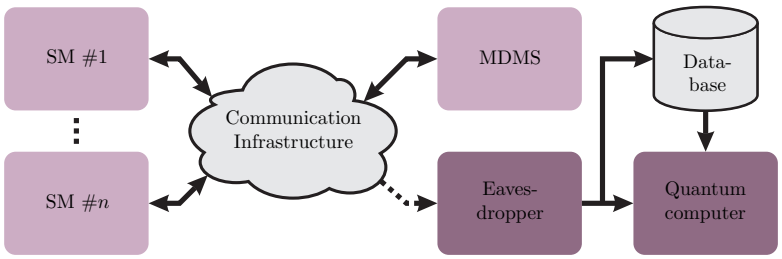


Figure 1. A typical security breach scenario in which an eavesdropper performs a sniffing attack and uses a quantum computer to decipher the SM data.

resistant asymmetric and symmetric cryptographic schemes within AMI—an architecture referred to as QR-AMI. Such integration is important for guaranteeing data security and protecting privacy in smart metering systems, especially in light of the emerging threat of quantum computational attacks. Critical research questions that deserve exploration in this context include: *What amount of hardware resources are requisite to facilitate the state-of-the-art PQC schemes within SMs and MDMS? Is it feasible to implement PQC schemes on processors with varying computational capacities, such as microcontrollers (low-power processor) and high-power processors for software-driven solutions or SoC devices designed for hardware/software co-design one?* The following sections elucidate these timely research questions.

3. A Quantum-Resistant AMI

A QR-AMI is designed to counter threats from adversaries with access to both quantum and classical computers. As such, QR-AMI must incorporate quantum-resistant asymmetric and symmetric cryptographic schemes. Figure 2 shows a block diagram illustrating the architecture of a QR-AMI. A key differentiation between QR-AMI and classical AMI is the cryptographic module employed: QR-AMI utilizes a QR-DCM that operates based on PQC schemes, whereas classical AMI incorporates a non-quantum-resistant Dedicated Cryptographic Module (DCM), which runs classical cryptographic schemes.

In a QR-AMI, any SM has access to PQC schemes. Consequently, a QR-DCM is required to interface with the SM to collect data and encrypt it using a quantum-resistant symmetric cryptographic scheme and send it through the AMI using data networks to MDMS. In this sense, QR-DCMs located on the consumer/prosumer side or on the electric utility side perform a key agreement through a Key Encapsulation Mechanism (KEM), generating a shared secret (i.e., a symmetric key), using a quantum-resistant asymmet-

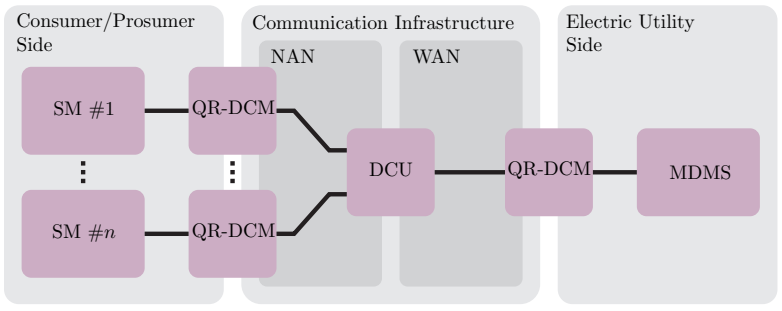


Figure 2. A block diagram of a QR-AMI, in which QR-DCM is the key element.

ric cryptographic scheme before encrypting and authenticating data. The QR-DCM at the consumer/prosumer side also has the capability of receiving encrypted data from the MDMS, which needs to be authenticated using Authenticated Encryption with Associated Data (AEAD), decrypted using the shared secret, and, finally, forwarded to the SM. In the model shown in Figure 2, there is one QR-DCM for each SM, although multiple SMs can be connected to one QR-DCM.

On the other hand, the QR-DCM on the electric utility side is implemented in the MDMS to send and receive encrypted and authenticated data. As the QR-DCM presented on the consumer or prosumer side, it checks the authenticity of received data through the AEAD before decrypting it using a quantum-resistant symmetric cryptographic scheme. Next, data correctly authenticated is forwarded to the MDMS. Naturally, a key agreement is previously executed using a quantum-resistant asymmetric cryptographic scheme. It is important to note that the QR-DCM on the electric utility side receives/sends a considerable amount of data depending on the size of the AMI (i.e., depending on how many SMs are presented on it). Therefore, the hardware resource of this QR-DCM must be carefully dimensioned for SMs and MDMS.

4. Cryptographic Schemes For a Quantum-Resistant DCM

QR-AMI uses asymmetric and symmetric cryptographic schemes to enable the secure key agreement between parties and the encryption/decryption of sensitive data. In this sense, Subsection 4.1 and Subsection 4.2 briefly describe the chosen quantum-resistant asymmetric and symmetric cryptography schemes, respectively.

4.1. Asymmetric Cryptographic Schemes

It is well-known that classical cryptographic schemes can perform a key agreement based on the hardness of factoring integers or computing discrete logarithms, such as RSA and ECC. Nonetheless, they are not quantum-resistant. With this concern in mind, the National Institute for Standards and Technology (NIST) started a PQC standardization process seeking to evaluate and select one or more quantum-resistant Public Key Encryption (PKE)/KEM and signature cryptographic schemes [Chen et al. 2016]. Among the submitted schemes, FrodoKEM [Alkim et al. 2020] and CRYSTALS-Kyber [Avanzi et al. 2021] schemes stood out from the others in the PKE/KEM category. The former is based on the Learning With Errors (LWE) problem proposed by Regev [Regev 2010], which is easily scalable, facilitating the adaptation of different security levels. Moreover, it is based on very conservative mathematical primitives due to the lack of any additional structure (i.e., ring structure), which provides, in theory, robust cryptographic principles despite degrading timing performance and requiring long keys and ciphertext. On the other hand, CRYSTALS-Kyber relies on the Module-Learning With Errors (M-LWE) problem, proposed by Langlois and Stehlé in [Langlois and Stehlé 2015], which is a generalization of the Ring-Learning With Errors (R-LWE) problem, proposed by Lyubashevsky in [Lyubashevsky et al. 2013]. Due to an additional structure, the R-LWE has opposite characteristics compared to the LWE. On the one hand, it considerably increases the performance and reduces the keys and ciphertext sizes, but on the other hand, it might be less resistant against malicious attacks.

M-LWE tries to get the best of the LWE and R-LWE. It has an additional structure but is less structured than R-LWE. According to recent cryptanalytic progress, a success-

ful attack against schemes relying on the M-LWE problem tends to be rarer than against schemes relying on the R-LWE [Bos et al. 2018]. It also performs similarly to the R-LWE and similar keys and ciphertext sizes. Finally, the M-LWE has better scalability than the R-LWE problem. Overall, we can say that the LWE problem is the most conservative, privileging consolidated principles, while the R-LWE is focused on performance. The M-LWE considers a trade-off between consolidated principles and performance.

FrodoKEM, relying on the LWE problem, remains as an alternate candidate in the NIST PQC standardization process in the PKE/KEM category and it has been still adopted by European authorities (e.g., German BSI and French ANSSI) due to its conservative approach. As requested in the NIST PQC standardization process, FrodoKEM is designed for Adaptive Chosen Ciphertext Attack (IND-CCA2) security at three levels: FrodoKEM-640, FrodoKEM-976, and FrodoKEM-1344, which aim to ensure a security level roughly equivalent to AES-128, AES-192, and AES-256 [Alkim et al. 2020], respectively. The CRYSTALS-Kyber, on the other hand, is the winner of the NIST PQC standardization process in the PKE/KEM category and has been standardized as ML-KEM in FIPS-203 [of Standards & Technology 2025]. It is also designed for IND-CCA2 security at three levels: CRYSTALS-Kyber-512, CRYSTALS-Kyber-976, and CRYSTALS-Kyber-1024 [Avanzi et al. 2021]. In this sense, FrodoKEM and CRYSTALS-Kyber schemes are considered for performing quantum-resistant KEM in QR-AMIs, and consequently, they are analyzed to integrate QR-DCM.

Table 1 summarizes the main characteristics of FrodoKEM and CRYSTALS-Kyber schemes. As mentioned, FrodoKEM relies on the LWE problem and the CRYSTALS-Kyber on the M-LWE problem. The type of the considered problem is reflected directly in the public key, private key, and ciphertext sizes, which are more than ten times larger in FrodoKEM than in CRYSTALS-Kyber. Considering the problem each scheme relies on, it can be said that the FrodoKEM scheme follows a conservative security approach, while the CRYSTALS-Kyber scheme is lightweight. In this sense, the CRYSTALS-Kyber scheme tends to consume fewer memory resources and can be significantly faster than the FrodoKEM scheme.

Moreover, Table 1 shows that the public key, private key, and ciphertext sizes of these PQC schemes are much larger than in classical cryptographic schemes. For instance, RSA usually employes keys with less than 4096-bits and ECC with less than 521-bits. Consequently, implementing PQC schemes in hardware- and memory-constrained devices is a sensitive issue; however, Tasopoulos *et al.* [Tasopoulos et al. 2023] showed that regarding performance, asymmetric quantum-resistant and classical cryptography demand almost the same energy consumption.

It is important to state that while this work adopts KEM for key agreement, it does not include any digital signature schemes, either classical or quantum-resistant. This design choice is deliberate and tied to our primary objective: Evaluating the feasibility of quantum-resistant KEMs and symmetric schemes within the hardware constraints of AMI devices. However, we recognize that the KEM alone does not authenticate the sender. In practice, digital signatures are essential to provide message origin authentication, particularly for control messages sent from utility providers to smart meters. This is a critical aspect of security in real-world deployments, as it prevents impersonation attacks and ensures non-repudiation. In future iterations of the QR-AMI, integrating post-quantum sig-

Table 1. Overview of FrodoKEM and CRYSTALS-Kyber schemes.

	FrodoKEM $640/976/1344$	CRYSTALS-Kyber $512/768/1024$
Class	Lattice	Lattice
Mathematical Problem	LWE	M-LWE
Public Key Size (bytes)	$9616/15632/21520$	$800/1184/1568$
Private Key Size (bytes)	$19888/31296/43088$	$1632/2400/3168$
Ciphertext Size (bytes)	$9720/15744/21632$	$768/1088/1568$
Approach	Conservative	Lightweight

nature schemes, such as ML-DSA, is crucial to achieving a complete quantum-resistant architecture.

4.2. Symmetric Cryptographic Schemes

A quantum computer will not significantly impact symmetric cryptography due to its mathematical principles. Consequently, all secure and standardized symmetric algorithms are potentially quantum-resistant. In addition, a suitable symmetric algorithm for AMIs must be reliable, fast, and lightweight. It is also desirable for this algorithm to provide AEAD. A suitable choice is, therefore, the AES, a standardized algorithm by the NIST in 2001 [Dworkin et al. 2001].

The AES is a block cipher of 128-bits, with three possible key sizes: 128, 192, and 256-bits. In this regard, the most conservative choice (i.e., 256-bit key) seems more appropriate in light of Grover's attack [Grassl et al. 2016]. Furthermore, the AES can be combined with the Galois/Counter Mode (AES-GCM), specified in NIST Special Publication $800-38$D [Dworkin 2007], for providing authenticated encryption (confidentiality and authentication). Therefore, the AES-256-GCM is considered and evaluated as the quantum-resistant symmetric cryptographic scheme for QR-DCMs. For the sake of simplicity, it will be called AES-GCM henceforth.

5. Experimental Implementation of QR-AMI

Upgrading the processing capacity of a MDMS is generally straightforward for electric utilities. In contrast, adapting existing SMs poses a significant challenge due to their limited processing capabilities. Hence, a fair implementation of QR-DCM becomes critical for ensuring quantum security in QR-AMI. Three primary approaches for implementing QR-DCM are:

- **Approach** $\#1$: It integrates the QR-DCM into an existing internal processor of SMs. Due to the limited computational capacity of current SMs, this approach

requires a comprehensive hardware redesign, including replacing the existing processor. This approach is best suited for SMs in the design phase.

- **Approach** $\#2$: It incorporates a specialized and auxiliary processor into the SM, exclusively for the execution of the QR-DCM. While this also requires significant hardware modifications, executing the cryptographic schemes on a specialized processor confers an additional layer of security. It is also advantageous for designing new SMs.
- **Approach** $\#3$: It employs a dedicated external module with its own processor to implement the QR-DCM. This module can easily interface with existing ports on SMs, thus providing an efficient avenue for implementing quantum-resistant cryptographic schemes in existing SMs. It is most appropriate for SMs already deployed in the field.

Although SMs can be significantly modified, such changes would require a rigorous re-certification process to ensure compliance with industry standards and national regulations, a task that is both time-consuming and costly. Considering the vast number of SMs already in deployment, **Approach** $\#3$ emerges as the most viable approach for fortifying existing AMI systems against both classical and quantum computational threats. The downside of this approach lies in the potential vulnerability of the serial communication between the dedicated external module and the SM, which can be mitigated by physically integrating the QR-DCM within the SM. However, in line with the Zero Trust principle, physical security alone is insufficient. A detailed discussion of this issue is beyond the scope of this paper.

By leveraging this approach, the QR-DCM offloads the cryptographic workload to an external module, avoiding extensive redesign of resource-constrained SMs. It should be highlighted that QR-DCM may be implemented on either a microcontroller (low-cost processor) or an SoC, each with distinct trade-offs. A microcontroller-based solution offers cost-efficiency at the expense of longer execution times, while an SoC-based solution, although more expensive, delivers hardware acceleration for compute-intensive tasks, significantly enhancing execution speed.

Concerning **Approach** $\#3$, Figure 3 shows the experimental implementation employed to simulate a QR-AMI utilizing QR-DCMs implemented on both specialized microcontroller and SoC development boards. Subsequent subsections will elaborate on the various aspects of this implementation.

5.1. Smart Meter

The SMs, tagged as $\#1A$ and $\#1$B in Figure 3, are SMW3000 models from WEG [WEG 2022], an ANSI bottom-connected three-phase meter for direct measurement up to 120A. This model supports the Device Language Message Specification (DLMS)/Companion Specification for Energy Metering (COSEM) communication protocol via various interfaces (e.g., local port, Ethernet, optical, and serial), allowing integration with multiple technologies. In this implementation, the SM communicates with the QR-DCM via serial DLMS/COSEM.

5.2. Quantum-Resistant DCM

The QR-DCMs, identified as $\#2A$, $\#2B$, and $\#2C$ in Figure 3, serve as critical elements for achieving quantum security in QR-AMI. These modules incorporate quantum-

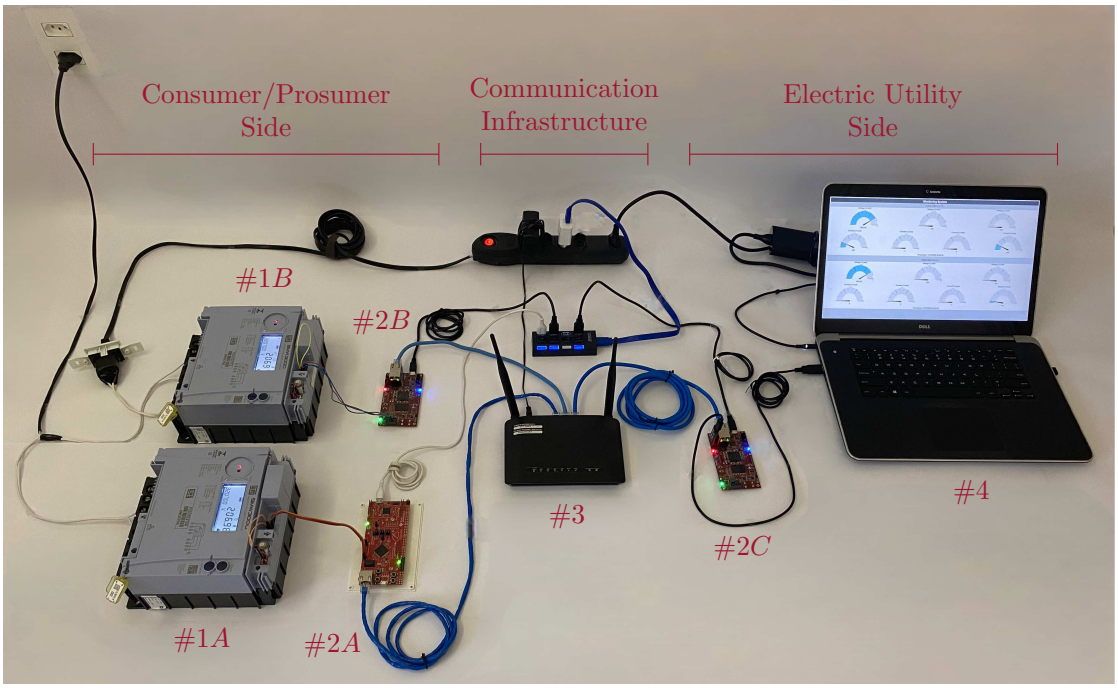


Figure 3. An experimental implementation of a QR-AMI.

resistant cryptographic schemes and the DLMS/COSEM communication protocol. The choice of processing power for the QR-DCM largely depends on the scale of the AMI. Two types of technologies are considered for implementing the QR-DCM, namely, microcontroller-based and SoC-based platforms:

- **Microcontroller** ($\#2A$): It relies on a device with low power consumption and, consequently, a not-so-fast processing unit for performing its tasks. It fits well for low processing power (e.g., processing a small amount of data). For instance, on the consumer/prosumer side, connected to only one or a few SMs, or even on the electric utility side when an AMI is dedicated to a small number of SMs. It uses the EK-TM4C129EXL kit [Instruments 2014], based on a 120 MHz Advanced RISC Machine (ARM) Cortex-M4F with 1024 kB flash, 256 kB SRAM, and 6 kB EEPROM. This platform supports only software implementations, resulting in longer execution times. Source code is available at [Lagrota 2024d].
- **SoC device** ($\#2B$ and $\#2C$): Recommended for high processing needs (e.g., many SMs), on either side of the AMI. It employs the MicroZed 7010 board [Xilinx 2018], which includes a Xilinx Zynq-7000 SoC featuring 1 GB DDR3 SDRAM, 128 Mb QSPI Flash, and an XC7Z010-1CLG400C chip. This chip integrates a Processing System (PS) (ARM Cortex-A9) and a Programmable Logic (PL) (Field Programmable Gate Array (FPGA)) with 28k logic cells, 17.6k Lookup Tables (LUTs), 35.2k registers, 60 Block Random Access Memory (BRAM), and 80 Digital Signal Processing (DSP) blocks. A 33.33 MHz oscillator feeds clocks to the PS and PL via a Phase-Locked Loop (PLL). This bsoard enables both software and hardware/software co-design implementations, with cryptographic routines accelerated in hardware. Implementations are available at [Lagrota 2024a, Lagrota 2024c, Lagrota 2024b].

Regardless of the platform, all QR-DCMs communicate with SMs via serial DLMS/COSEM and with the MDMS using JSON-formatted serial communication. Ethernet is used to interface with the Data Concentrator Unit (DCU), ensuring interoperability across different AMI components.

5.3. Communication Infrastructure

The communication infrastructure only comprises a commercial D-Link DSL-2740E router, tagged as $\#3$ in Figure 3, which forwards packets from one node to another (i.e., allows two-way communication) using the Ethernet network. Note that all the QR-DCMs are connected in the same data network, so they can easily communicate. As we aim to focus on the implementation aspects of the quantum-resistant functionality in a QR-AMI, the type of communication infrastructure is not a relevant issue; therefore, no further efforts are necessary to implement the communication infrastructure.

5.4. MDMS

The MDMS, tagged as $\#4$ in Figure 3, is emulated on a Python application with Flask, a micro-framework suitable for small applications with simple requirements, communicating with the QR-DCM. Mainly, it receives decrypted and authenticated data from the QR-DCMs, which contain information collected by SMs, and presents it on a screen using Flask. Similar to the communication infrastructure, no further efforts were spent trying to emulate an MDMS because it would not bring any relevant contribution to the scope of this work.

6. Experimental Results

This section analyzes the experimental implementation of a QR-AMI, focusing on the QR-DCM as the key enabler for quantum security in the AMI. Quantum-resistant asymmetric (FrodoKEM and CRYSTALS-Kyber) and symmetric (AES-GCM) cryptographic schemes are implemented on two different platforms as described in Subsection 5.2, using the following implementations:

- Impl. $\#1$: A software-only implementation executed on the ARM Cortex-M4F in the EK-TM4C129EXL.
- Impl. $\#2$: A software-only implementation using the ARM Cortex-A9 on the MicroZed 7010 board. The ARM Cortex-A9 is a high-power processor.
- Impl. $\#3$: A hardware/software co-design implementation leveraging both the ARM Cortex-A9 and the FPGA on the MicroZed 7010 board (i.e., SoC device).

Impl. $\#1$ and $\#2$ aim to contrast the performance capabilities between low- and high-power processing units that rely solely on a microcontroller (ARM Cortex-M4F) and a more potent processor (ARM Cortex-A9), respectively. Conversely, Impl. $\#3$ aims to quantify the benefits of hardware acceleration. Notice that hardware acceleration for symmetric cryptography is not considered in this analysis, as its impact is deemed negligible, which is substantiated in the subsequent discussions.

To ensure a fair comparison, the source codes for all the implementations are adapted from those submitted to the NIST PQC standardization process [Chen et al. 2016]. Specifically, regarding FrodoKEM, only the FrodoKEM-640 version is considered for hardware acceleration due to the excessive resource consumption, particularly

BRAMs, of its more secure variants (i.e., FrodoKEM-976 and FrodoKEM-1344) [da Costa et al. 2022b]. For CRYSTALS-Kyber, we focus on CRYSTALS-Kyber-512 to make a fair security-level comparison with FrodoKEM.

Moreover, asymmetric and symmetric cryptographic scheme implementations regarding resource usage, timing performance, and energy consumption are detailed in Subsections 6.1, 6.2, and 6.3, respectively. Lastly, Subsection 6.4 summarizes this section with a few comments.

6.1. Resource Usage

This subsection analyzes the resource usage of Impl. #1, #2, and #3. In this sense, Subsection 6.1.1 focuses on memory resources and consequently pays attention to the software aspects, whereas Subsection 6.1.2 addresses the hardware resources from FPGA, emphasizing the hardware/software co-design.

6.1.1. Memory Resources

This analysis assesses the memory resource usage for the software components of Impl. #1 and #2, as well as the software aspect of Impl. #3. Three primary memory segments are scrutinized for each configuration: (i) the text segment, addressing the executable instructions; (ii) the read-only data segment, storing immutable constants such as global constants; and (iii) the data segment, containing initialized static variables, which are mutable at runtime. The text and read-only data segments usually reside in flash memory because they are unchangeable, whereas the editable data segment is stored in RAM.

Figure 4 illustrates the memory consumption across these three segments. It is evident that the ARM Cortex-M4F-based Impl. #1 utilizes fewer memory resources compared to the ARM Cortex-A9-based Impl. #2. Although the source codes across these implementations are functionally equivalent, the compiled binaries exhibit discrepancies in resource utilization. Platforms with higher computational power, like the ARM Cortex-A9, inherently entail more complexity, particularly in background management processes, thus leading to elevated memory consumption. These platforms are, however, endowed with more abundant resources to accommodate this added complexity, as elaborated in Section 5.2.

A direct comparison between Impl. #2 and #3, both based on the ARM Cortex-A9, reveals that the hardware/software co-design approach (Impl. #3) is more memory-efficient. This efficiency stems from the fact that functions executed in hardware do not require compilation, thereby conserving memory. Nonetheless, these savings are marginal, attributed to the partial hardware acceleration of certain functions, as elaborated in [da Costa et al. 2022b, da Costa et al. 2022a].

Interestingly, within the same implementation environment, FrodoKEM is found to be more memory-efficient than CRYSTALS-Kyber in Impl. #2 and #3. This trend is inverted in Impl. #1, where CRYSTALS-Kyber demonstrates greater memory efficiency, except in the read-only data segments, where the sizes are nearly identical. Despite the similar source codes for both cryptographic schemes, the observed discrepancy is due to differences in the compiler technologies employed. As such, the memory utilization characteristics under one compiler configuration may not directly extend to another.

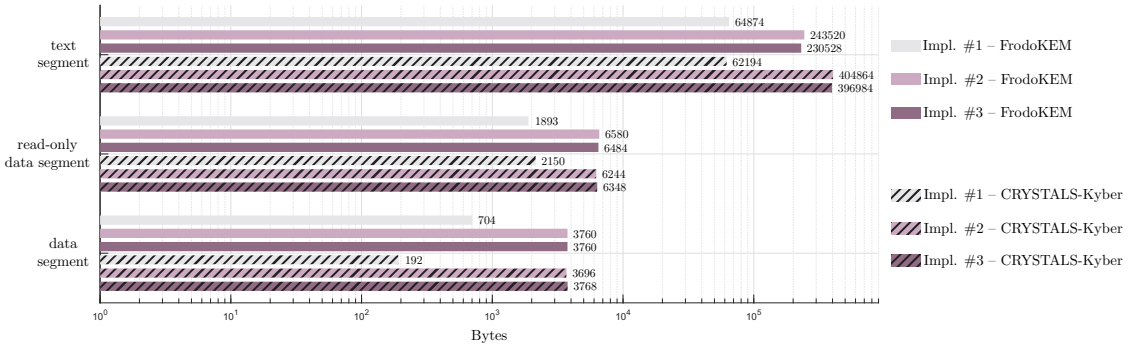


Figure 4. Memory usage for text, read-only data, and data segments in bytes of the FrodoKEM and CRYSTALS-Kyber schemes for Impl. $\#1$, $\#2$, and $\#3$.

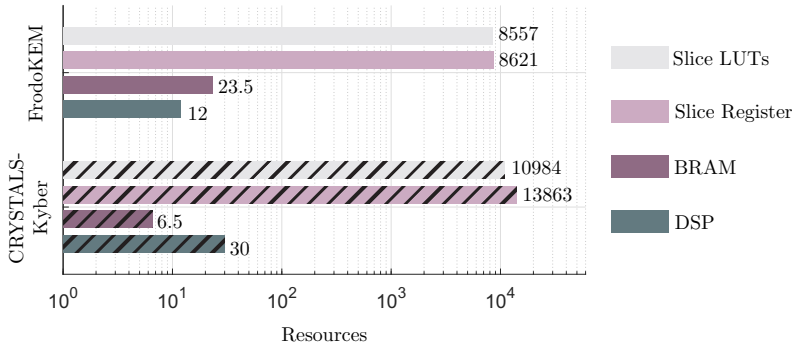


Figure 5. Hardware resource usage of the FrodoKEM and CRISTALS-Kyber schemes in terms of Slice LUTs, Slice Register, BRAM, and DSP blocks.

6.1.2. Hardware resources

This analysis compares the hardware resource usage of the FrodoKEM and CRYSTALS-Kyber schemes, and therefore, it considers only Impl. $\#3$. Four main resources are evaluated: (i) slice LUTs, which are small asynchronous RAMs manipulated to implement combinational logic; (ii) slice registers, based on Flip Flop (FF) and used to hold states; (iii) BRAM, based on synchronous RAM and used to store large volumes of data; and (iv) DSP blocks, which are dedicated blocks for performing multiplications, increasing performance, and reducing logic use (e.g., slice LUTs and registers).

Figure 5 shows the hardware resource usage of the FrodoKEM and CRISTALS-Kyber. It reveals that CRYSTALS-Kyber consumes more slice LUTs, slice registers, and DSP blocks than FrodoKEM, owing to its more complex hardware functions [da Costa et al. 2022b, da Costa et al. 2022a]. Conversely, FrodoKEM uses more BRAM due to its longer key lengths, as detailed in Table 1.

6.2. Timing Analysis

This subsection presents a timing analysis. In this regard, Subsection 6.2.1 compares the execution time of the asymmetric cryptographic schemes, while Subsection 6.2.2 presents the timing analysis of the symmetric cryptography.

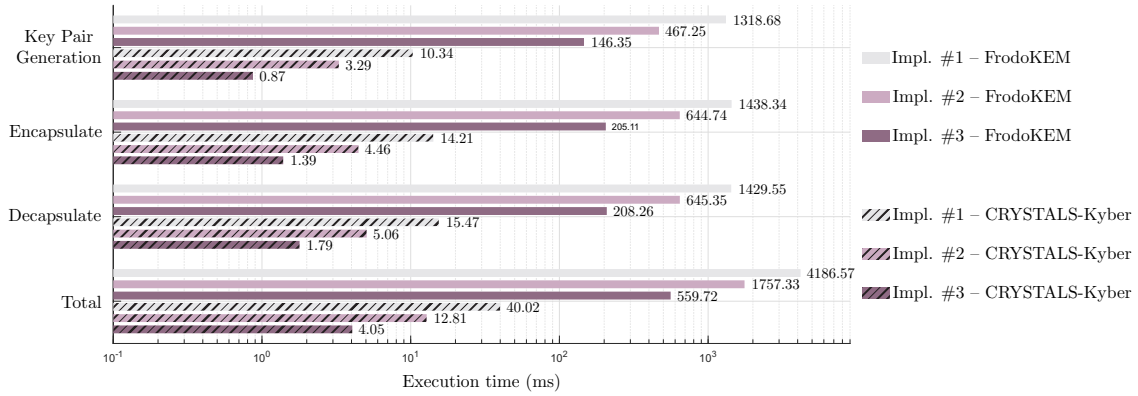


Figure 6. Timing performance in ms of the FrodoKEM and CRYSTALS-Kyber schemes using Impl. $\#1$, $\#2$, and $\#3$.

6.2.1. Asymmetric cryptography

It provides an in-depth evaluation of the execution times associated with key pair generation, encapsulation, and decapsulation for the FrodoKEM and CRYSTALS-Kyber schemes, as implemented using Impl. $\#1$, $\#2$, and $\#3$.

Figure 6 shows the total execution times for the aforementioned cryptographic schemes. The performance disparity between the FrodoKEM and CRYSTALS-Kyber is significant. Regardless of the implementation method, CRYSTALS-Kyber consistently exhibits a performance that is at least a hundredfold superior to that of FrodoKEM. These empirical results substantiate the theoretical comparisons presented in Subsection 4.1.

Concerning FrodoKEM, the hardware-accelerated Impl. $\#3$ significantly outperforms its solely software-based counterparts. Specifically, it is around 7.5 and 3.1 times faster than Impl. $\#1$ and $\#2$, respectively. This substantially reduces total execution time, decreasing from 4186.57 ms and 1757.33 ms to a 559.72 ms when hardware acceleration is leveraged.

Similarly, for the CRYSTALS-Kyber scheme, Impl $\#3$ shows superior performance, being approximately 9.9 and 3.2 times faster than Impl. $\#1$ and $\#2$, respectively. Consequently, the total execution time for Impl. $\#1$ and $\#2$ is reduced from 40.02 ms and 12.81 ms to an impressive 4.05 ms when utilizing hardware acceleration, as in Impl. $\#3$.

6.2.2. Symmetric cryptography

In symmetric cryptography, the execution time needed for data encryption, decryption, and data authentication is closely tied to the data length in the AES-GCM scheme. For instance, to encrypt, decrypt, and authenticate a 50-byte data packet, Impl. $\#1$ and $\#2$ require 0.262 ms and 0.062 ms, respectively. These results show that the execution time for symmetric cryptography is negligible compared to asymmetric cryptography. Hence, implementing hardware acceleration for AES-GCM would not yield substantial time-saving benefits for small data packets, as typically encountered in AMIs. On the other hand, such hardware acceleration would consume additional hardware resources. Consequently, symmetric cryptography in this experiment is confined to a software implementation.

Table 2. Energy consumption between FrodoKEM and CRYSTALS-Kyber

Scheme	Impl.	Time (ms)	Power (W)	Energy (mJ)
FrodoKEM	Impl. #1	4186.57	0.35	1454.79
	Impl. #2	1757.33	1.65	2903.11
	Impl. #3	559.72	2.05	1147.43
CRYSTALS-Kyber	Impl. #1	40.02	0.35	13.91
	Impl. #2	12.81	1.65	21.16
	Impl. #3	4.05	1.83	7.43

6.3. Energy Consumption

This subsection analyzes the energy consumption patterns. It employs timing estimates delineated in Subsection 6.2.1 and the energy consumption for key pair generation, encapsulation, and decapsulation. Power consumption estimate for Impl. $\#1$ is extracted from the EK-TM4C129EXL datasheet [Instruments 2014]. For Impl. $\#2$ and $\#3$, power consumption estimates are furnished by the synthesizer of the SoC device. Note that these estimates include activating the communication layer, which enables data exchange between QR-DCMs and contributes to significant power consumption.

Table 2 lists the energy consumption of FrodoKEM and CRYSTALS-Kyber. First, as FrodoKEM is considerably slower than CRYSTALS-Kyber while having a similar power estimate for each implementation, the former consumes much more energy. Second, Impl. $\#1$ consumes much less power than Impl. $\#2$ and $\#3$ which reflects in its performance. However, no significant difference exists between Impl. $\#2$ and $\#3$, indicating that the hardware acceleration is energy efficient. Lastly, we see that the Impl. $\#3$ is the most energy efficient, followed by Impl. $\#1$. Overall, the most sophisticated hardware (i.e., SoC device) attains the lowest energy consumption, indicating its feasibility for QR-AMIs.

6.4. General Comments

In summary, the choice of cryptographic scheme and hardware platform for QR-DCM should be based on the specific requirements of the QR-AMI. Regarding cryptographic schemes, FrodoKEM, despite its higher computational requirements, is preferable when robust cryptographic principles based on well-established mathematical foundations are important, as discussed in Section 4.1. However, its slower execution times could be a bottleneck in large-scale QR-AMI deployment, potentially failing to meet timing constraints. On the other hand, CRYSTALS-Kyber offers a more efficient alternative for most scenarios. As for symmetric cryptography, AES-GCM appears universally suitable, being lightweight and efficient enough to be implemented on any platform, as noted in Subsection 4.2.

In the context of hardware choices for QR-DCM, the scenarios best suited for microcontroller-based hardware (i.e., Impl. $\#1$ and $\#2$) and SoC devices (i.e., Impl. $\#3$) differ significantly. If the computational demands are low, such as in cases where the QR-DCM is interfacing with only a single SM, then a microcontroller-based implemen-

tation (Impl. $\#1$) would be more cost-effective. For moderate to high computational demands, such as when multiple SMs are connected, an ARM Cortex-A9-based implementation (Impl. $\#2$) would be appropriate. For scenarios demanding extremely high computational capabilities, Impl. $\#3$ would be the logical choice.

From the perspective of electric utilities, the scalability of the QR-AMI should be a key factor in deciding the appropriate implementation. Small-scale QR-AMIs may manage with Impl. $\#1$, but as the size and complexity grow, transitioning to Impl. $\#2$ or even Impl. $\#3$ would become increasingly necessary to meet timing, computational requirements, and energy consumption.

7. Conclusions

This paper has discussed QR-AMI for securing sensitive data against attacks from quantum and classical computers. The QR-AMI mainly relies on QR-DCMs, which implement an asymmetric quantum-resistant cryptographic scheme, one with more robust cryptographic principles (i.e., FrodoKEM) or another with less computational burden (i.e., CRYSTALS-Kyber). In addition, QR-DCMs implement a symmetric cryptographic scheme that efficiently encrypts, decrypts, and authenticates data (i.e., AES-256-GCM). It also has shown that QR-DCM in a microcontroller suits well in SMs with less processing power. On the other hand, SoC device is necessary for MDMSs that exchange a large volume of data. Overall, QR-DCMs based on external and dedicated hardware are suitable for making the existing SMs quantum-resistant.

References

[Ahn et al. 2022] Ahn, J. et al. (2022). Toward quantum secured distributed energy resources: Adoption of post-quantum cryptography (PQC) and quantum key distribution (QKD). *Energies*, 15(3). pages 2

[Alkim et al. 2020] Alkim, E. et al. (2020). FrodoKEM: Learning with errors key encapsulation. Technical report, NIST, Gaithersburg, MD. pages 5, 6

[Avanzi et al. 2021] Avanzi, R. et al. (2021). CRYSTALS-Kyber algorithm specifications and supporting documentation. Technical report, NIST. pages 5, 6

[Barker et al. 2024] Barker, W., Polk, W., and Souppaya, M. (2024). NIST IR 8547 — transition to post-quantum cryptography standards. Technical report, NIST. pages 2

[Borges et al. 2020] Borges, F., Reis, P. R., and Pereira, D. (2020). A comparison of security and its performance for key agreements in post-quantum cryptography. *IEEE Access*, 8:142413–142422. pages 2

[Bos et al. 2018] Bos, J. et al. (2018). CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM. In *Proc. IEEE Eur. Symp. Security and Privacy*, pages 353–367. pages 6

[Bui et al. 2012] Bui, N., Castellani, A. P., Casari, P., and Zorzi, M. (2012). The internet of energy: A web-enabled smart grid system. *IEEE Netw.*, 26(4):39–45. pages 3

[Chen et al. 2016] Chen, L. et al. (2016). Report on post-quantum cryptography. Technical Report 8105, NIST, Gaithersburg, MD. pages 5, 10

[Cheng et al. 2019] Cheng, C., Qin, Y., Lu, R., Jiang, T., and Takagi, T. (2019). Batten down the hatches: Securing neighborhood area networks of smart grid in the quantum era. *IEEE Trans. Smart Grid*, 10(6):6386–6395. pages 2

[da Costa et al. 2022a] da Costa, V. L. R., Camponogara, Â., López, J., and Ribeiro, M. V. (2022a). The feasibility of the CRYSTALS-Kyber scheme for smart metering systems. *IEEE Access*, 10:131303–131317. pages 2, 11, 12

[da Costa et al. 2022b] da Costa, V. L. R., López, J., and Ribeiro, M. V. (2022b). A system-on-a-chip implementation of a post-quantum cryptography scheme for smart meter data communications. *Sensors*, 22(19). pages 2, 11, 12

[Dworkin 2007] Dworkin, M. J. (2007). *Sp 800-38d. recommendation for block cipher modes of operation: Galois/counter mode (GCM) and GMAC*. National Institute of Standards & Technology. pages 7

[Dworkin et al. 2001] Dworkin, M. J., Barker, E., Nechvatal, J., Foti, J., Bassham, L., Roback, E., and Dray, J. (2001). Advanced encryption standard (AES). pages 7

[Ghasempour 2019] Ghasempour, A. (2019). Internet of things in smart grid: Architecture, applications, services, key technologies, and challenges. *Inventions*, 4(1):22. pages 1

[Ghasempour and Lou 2017] Ghasempour, A. and Lou, J. (2017). Advanced metering infrastructure in smart grid: Requirements, challenges, architectures, technologies, and optimizations. In *Smart Grids: Emerging Technologies, Challenges and Future Directions*. Nova Science Publishers. pages 1

[Grassl et al. 2016] Grassl, M., Langenberg, B., Roetteler, M., and Steinwandt, R. (2016). Applying Grover's algorithm to AES: Quantum resource estimates. In Takagi, T., editor, *Post-Quantum Cryptography*, pages 29–43, Cham. Springer International Publishing. pages 7

[Instruments 2014] Instruments, T. (2014). Tiva™ TM4C129ENCPDT Microcontroller. pages 9, 14

[Lagrota 2024a] Lagrota, V. (2024a). Frodo FPGA. `https://gitlab.com/vinicius.lagrota/frodo-fpga`. pages 9

[Lagrota 2024b] Lagrota, V. (2024b). IP-repository. `https://gitlab.com/vinicius.lagrota/ip-repository`. pages 9

[Lagrota 2024c] Lagrota, V. (2024c). Kyber FPGA. `https://gitlab.com/vinicius.lagrota/kyber-fpga`. pages 9

[Lagrota 2024d] Lagrota, V. (2024d). PQC TM4C129 - ARM M4F. `https://gitlab.com/vinicius.lagrota/pqc-tm4c129`. pages 9

[Langlois and Stehlé 2015] Langlois, A. and Stehlé, D. (2015). Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75(3):565–599. pages 5

[Li et al. 2017] Li, S., Xue, K., Yang, Q., and Hong, P. (2017). PPMA: Privacy-preserving multisubset data aggregation in smart grid. *IEEE Trans. Ind. Inform.*, 14(2):462–471. pages 1

[Lyubashevsky et al. 2013] Lyubashevsky, V., Peikert, C., and Regev, O. (2013). On ideal lattices and learning with errors over rings. *J. ACM*, 60(6). pages 5

[Mohassel et al. 2014] Mohassel, R. R., Fung, A., Mohammadi, F., and Raahemifar, K. (2014). A survey on advanced metering infrastructure. *Int. J. Electr. Power Energy Syst.*, 63:473–484. pages 3

[Naderi and Asrari 2023] Naderi, E. and Asrari, A. (2023). A remedial action scheme to mitigate market power caused by cyberattacks targeting a smart distribution system. *IEEE Trans. Ind. Inform.*, pages 1–12. early access, doi: 10.1109/TII.2023.3304049. pages 1

[of Standards & Technology 2025] of Standards & Technology, N. I. (2025). *Module-Lattice-Based Key-Encapsulation Mechanism Standard*. National Institute of Standards & Technology. pages 6

[Regev 2010] Regev, O. (2010). The learning with errors problem (invited survey). In *Proc. IEEE Annu. Conf. on Computational Complexity*, pages 191–204. pages 5

[Shor 1994] Shor, P. W. (1994). Algorithms for quantum computation: Discrete logarithms and factoring. In *Proc. 35th Annu. Symp. Foundations of Computer Science*, pages 124–134. pages 2

[Tasopoulos et al. 2023] Tasopoulos, G., Dimopoulos, C., Fournaris, A. P., Zhao, R. K., Sakzad, A., and Steinfeld, R. (2023). Energy consumption evaluation of post-quantum TLS 1.3 for resource-constrained embedded devices. Cryptology ePrint Archive, Paper 2023/506. `https://eprint.iacr.org/2023/506`. pages 6

[Wang and Lu 2013] Wang, W. and Lu, Z. (2013). Cyber security in the smart grid: Survey and challenges. *Comput. Netw.*, 57(5):1344 – 1371. pages 1, 2

[WEG 2022] WEG (2022). Intelligent power meters – SMW series. pages 8

[Xilinx 2018] Xilinx (2018). Zynq-7000 SoC Data Sheet: Overview. pages 9

Aplicação prática da quebra do KeeLoq: uma abordagem de baixo custo

Edison M. Junior, Vinícius Lagrota[1]

[1] Centro de Pesquisa e Desenvolvimento para a Segurança das Comunicações (CEPESC)
Brasília, DF – Brazil.

{junior170287@gmail.com, vinicius.lagrota@gmail.com

Abstract. *This work presents a low-cost attack against the HCS201 integrated circuit, widely used in access control systems based on KeeLoq. Through differential power analysis, it was possible to extract the cryptographic key stored in the device, demonstrating the feasibility of the attack without the need for high-cost measurement equipment. The experiments showed that the manufacturers analyzed use the same cryptographic key across all their devices, making them highly vulnerable to large-scale cloning and piracy attacks. As a possible mitigation strategy, the adoption of key derivation mechanisms, such as HKDF, is proposed to generate unique keys for each transmitter, preventing the extraction of a single key from compromising the entire system.*

Resumo. *Este trabalho apresenta um ataque de baixo custo ao circuito integrado HCS201, amplamente utilizado em sistemas de controle de acesso baseados no KeeLoq. Por meio da análise de potência diferencial, foi possível extrair a chave criptográfica armazenada no dispositivo, demonstrando a viabilidade do ataque sem a necessidade de equipamentos de medição de alto custo. Os experimentos mostraram que os fabricantes analisados utilizam a mesma chave criptográfica em todos os seus dispositivos, tornando-os altamente vulneráveis a ataques de clonagem e pirataria em larga escala. Como possível estratégia de mitigação, é proposto a adoção de mecanismos de derivação de chaves, como o HKDF, para gerar chaves únicas para cada transmissor, impedindo que a extração de uma única chave comprometa todo o sistema.*

1. Introdução

A proteção de informações e a garantia de comunicações seguras tornaram-se aspectos centrais na era digital, impulsionados pela crescente demanda por privacidade, integridade e confiabilidade dos dados em diversos setores [Sun et al. 2025]. Nesse cenário, a criptografia assume papel fundamental ao oferecer mecanismos robustos que asseguram a confidencialidade de informações sensíveis e a autenticidade de transações, tanto em ambientes corporativos quanto em dispositivos pessoais. Com a popularização de sistemas conectados e a expansão da Internet das Coisas (IoT), tornou-se comum a adoção de soluções criptográficas em sistemas embarcados [Thabit et al. 2023]. Esses sistemas, por operarem com recursos computacionais limitados, requerem algoritmos leves, de baixo consumo de energia e com segurança suficiente para proteger dados críticos [Soto-Cruz et al. 2024].

Entretanto, a limitação de recursos e a pressão por custos baixos frequentemente resultam em implementações criptográficas vulneráveis [Ji et al. 2023]. Consequentemente, sistemas embarcados tornam-se alvos comuns de ataques, especialmente os chamados ataques de canal lateral, do inglês *side-channel attack*s (SCAs) [Kocher 1996]. Tais ataques exploram características físicas da execução, como consumo de potência, tempo ou emissão eletromagnética, para recuperar segredos criptográficos. Dentre esses, destaca-se a análise de potência diferencial, do inglês *differential power analysis* (DPA), que aplica técnicas estatísticas sobre traços de energia para inferir valores sensíveis, como chaves criptográficas [Kocher et al. 1999].

Nesse contexto, o algoritmo KeeLoq [Marneweck 1996], desenvolvido para sistemas com restrições de hardware, é amplamente utilizado em controles remotos de portões e sistemas automotivos. Sua popularidade decorre da simplicidade e eficiência, mas, justamente por isso, sua implementação em larga escala torna-se vulnerável à extração de chaves criptográficas. Uma chave comprometida representa ameaça direta à segurança física, permitindo que atacantes obtenham acesso indevido a propriedades privadas, comprometendo o objetivo central de sistemas como os de controle de acesso.

Estudos anteriores demonstraram que é possível quebrar o KeeLoq utilizando SCAs [Paar et al. 2009], mas essas abordagens dependem de equipamentos sofisticados e de alto custo, como osciloscópios de alta taxa de amostragem, dificultando a replicação dos experimentos e limitando seu uso em cenários reais. Isso motiva a busca por alternativas mais acessíveis. Neste trabalho, propõe-se uma abordagem de baixo custo para explorar vulnerabilidades do KeeLoq embarcado nos chips da família HCS da Microchip, utilizando a técnica de DPA. As principais contribuições deste estudo são: i) a descrição da vulnerabilidade explorável no KeeLoq; ii) a implementação prática do ataque com dispositivos de baixo custo; e iii) a discussão de possíveis estratégias de mitigação.

Este artigo está organizado da seguinte forma: a Seção 2 apresenta os trabalhos relacionados; a Seção 3 discute os conceitos necessários para compreensão do estudo; a Seção 4 descreve a análise teórica do ataque; a Seção 5 detalha a implementação prática e discute possíveis mitigações; e a Seção 6 apresenta as conclusões finais.

2. Trabalhos relacionados

Diversos ataques ao KeeLoq podem ser encontrados na literatura. Os primeiros ataques foram publicados por Bogdanov [Bogdanov 2007]. Um dos ataques, baseado na técnica *slide, guess-and-determine* e ataques lineares, precisa de $2^{50.6}$ cifrações do KeeLoq. Já um segundo ataque, mais eficiente, necessita de 2^{37} cifrações. A desvantagem desses ataques é a necessidade de possuir o *codebook* completo, ou seja, ter acesso aos 2^{32} pares de claros-cifrados. Os resultados apresentados por Bogdanov já traziam indícios de severas vulnerabilidades no KeeLoq. Posteriormente, Courtois *et al.* [Courtois et al. 2008] propôs outros ataques ao KeeLoq utilizando somente a técnica *slide*. Em um dos ataques, Courtois *et al.* necessitou de $2^{51.4}$ cifrações do KeeLoq e 2^{16} pares de claros-cifrados. Em seu segundo ataque, este necessitando de quase todo o *codebook*, foi possível revelar a chave secreta usando 2^{27} cifrações. Já Indesteege *et al.* [Indesteege et al. 2008] apresentou um ataque prático ao KeeLoq, no qual é capaz de recuperar a chave secreta com complexidade de $2^{44.5}$ cifrações utilizando ataques de *slide* e *meet-in-the-middle*. Em um ataque, usando poucos pares de claro-cifrado do *codebook*, a chave secreta pode ser re-

cuperada em menos de 8 dias em CPUs com 64 núcleos. Já usando um *codebook* de 2^{16} pares de claro-cifrado, é capaz de executar o ataque em menos de 4 dias no mesmo *setup*.

Apesar de avanços consideráveis, os ataques citados são aplicados somente ao KeeLoq identification friend or foe (IFF), mas não são aplicáveis ao modo *Code Hopping*. De forma resumida, o IFF é um sistema de autenticação bidirecional, onde um dispositivo envia um desafio criptográfico e aguarda uma resposta única gerada a partir de uma chave secreta, garantindo maior segurança contra ataques de repetição. Já o modo *Code Hopping* é um mecanismo de autenticação unidirecional, onde cada transmissão usa um código variável gerado por um algoritmo, impedindo ataques de captura e repetição de códigos, sendo amplamente utilizado em controles remotos de veículos e portões. Neste sentido, devido à impossibilidade de ataques de captura e repetição, os ataques acima ficam inviáveis devido à impossibilidade de construir um *codebook*, já que o adversário terá acesso apenas ao cifrado, neste caso. Tendo em vista que os produtos comerciais utilizam o KeeLoq no modo *Code Hopping*, como, por exemplo, os chips da família HCS da Microchip, os ataques acima não são possíveis de serem aplicados.

No entanto, [Eisenbarth et al. 2008] propôs um ataque ao KeeLoq no modo *Code Hopping* em chips da famíla HCS utilizando SCA, mais especificamente, DPA. Combinando DPA com características específicas do KeeLoq, foi possível revelar de forma eficiente a chave secreta armazenada no dispositivo. Ainda foi mostrado como deve ser o procedimento para clonar um dispositivo e como realizar uma negação de serviço (DoS) em sistemas de controle de acesso com usam o KeeLoq. Apesar do trabalho de Eisenbarth *et al.* quebrar completamente o KeeLoq, foram necessário a utilização de equipamentos de alto custo e de difícil mobilidade. Neste sentido, uma lacuna notável na literatura existente diz respeito a soluções igualmente eficazes à apresentada por Eisenbarth *et al.*, mas que necessitam de um custo significantemente menor do que o requerido no estudo mencionado. Portanto, o presente trabalho busca preencher esta lacuna apresentando um método de baixo custo para a quebra do KeeLoq.

3. Fundamentos

Esta seção apresenta de forma detalhada o Rolling Code, técnica utilizada no KeeLoq. Além disso, os principais aspectos do KeeLoq são discutidos.

3.1. A tecnologia Rolling Code

A tecnologia Rolling Code (código rotativo), é usada principalmente em dispositivos de controle remoto, como alarmes automotivos e portões eletrônicos, trazendo maior nível de segurança por evitar que códigos de acesso sejam interceptados e reutilizados por terceiros. Essa tecnologia baseia-se em criptografia simétrica e é fundamentalmente simples uma vez que é utilizada em dispositivos de baixo poder de processamento e de baixo custo. No funcionamento dessa tecnologia apenas um dispositivo transmite (o transmissor) e apenas o outro dispositivo recebe (receptor) a informação. Cada vez que o transmissor efetua uma transmissão, ele gera um código sequencial que é transmitido de forma criptografada. Uma vez que esse código é recebido pelo receptor, este tem a capacidade de avaliar se tal código é válido baseando no código anteriormente transmitido. Se um código antigo for interceptado e utilizado novamente, ele será automaticamente rejeitado, uma vez que cada código só é aceito uma única vez.

3.1.1. O algoritmo KeeLoq

O KeeLoq é um algoritmo criptográfico simétrico com chave de 64 bits e bloco de 32 bits, desenvolvido para a tecnologia Rolling Code, garantindo que cada transmissão entre o transmissor e o receptor seja única e segura, protegendo contra ataques de clonagem ou interceptação. Esse mecanismo é amplamente utilizado em sistemas de controle de acesso, como anti-furto veicular e abridores de portas de garagem. No funcionamento do sistema, o transmissor contém um codificador e o receptor, um decodificador, ambos compartilhando uma chave secreta. Além disso, a comunicação é sincronizada por um contador de 16 bits, que é incrementado a cada nova transmissão do código variável [Eisenbarth et al. 2008].

A função de cifração do KeeLoq, ilustrada na Figura 1, pode ser representada matematicamente como uma permutação bijetiva sobre um espaço de blocos de 32 bits $E_K : \{0,1\}^{32} \rightarrow \{0,1\}^{32}$, tal que E_K representa a função de cifração parametrizada por uma chave secreta K de 64 bits. Isso significa que E_K recebe uma entrada P (um bloco de 32 bits) e produz uma saída C (outro bloco de 32 bits), garantindo que cada entrada tenha exatamente uma saída correspondente (função bijetiva): $C = E_K(P)$, com $P, C \in \{0,1\}^{32}$.

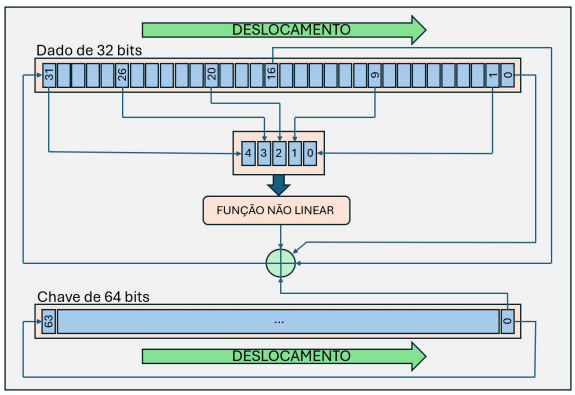


Figura 1. Função criptográfica de cifração do KeeLoq.

Após a inserção da chave e do valor em claro, são executados 528 ciclos de processamento, ao final dos quais o bloco de dados conterá o valor cifrado. Em cada ciclo de execução, tanto o barramento de dados quanto a chave são deslocados em um bit para a direita. Simultaneamente, os bits de posição 31, 26, 20, 9 e 1 do bloco de dados são utilizados como entrada para a função não-linear, do inglês *non-linear function* (NLF), que produz um único bit de saída. No mesmo ciclo, os bits de posição 16 e 0 do bloco de dados, combinados com o bit de saída da NLF e o bit 0 da chave, são processados por meio de uma operação XOR. O resultado dessa operação determina o bit que será inserido na posição 31 do barramento de dados após o deslocamento. Além disso, o bit 0 da chave é realocado para a posição 63 após o deslocamento, criando um efeito de rotação da chave a cada ciclo.

O processo de decifração segue a lógica inversa da cifração. Os 32 bits do bloco cifrado são inseridos na posição correspondente aos dados, e a mesma chave de 64 bits utilizada na cifração é carregada em sua posição original. Para recuperar os dados originais,

todas as operações realizadas na cifração são revertidas, aplicando as funções opostas em cada etapa, resultando na restauração do bloco de dados original.

3.2. O circuito integrado HCS201

O circuito integrado HCS201 [Microchip 2001] faz parte de uma linha de chips desenvolvidos pela Microchip Technology e é amplamente utilizado em sistemas de controle de acesso e segurança. Esse chip disponibiliza uma implementação da tecnologia Rolling Code e faz uso do algoritmo KeeLoq. Além disso, ele é adequado para aplicações de baixo custo, pois a criptografia é realizada diretamente pelo chip, além de gerar um sinal de saída próprio para ser aplicado a um transmissor de radiofrequência.

O chip contém internamente uma chave de 64 bits, um número serial de 28 bits e um contador de 16 bits. Inicialmente, o chip é fabricado sem a chave e o número serial, permitindo que o fabricante do dispositivo criptográfico grave essas informações durante a configuração. Para isso, o chip dispõe de uma memória não-volátil do tipo EEPROM, onde são armazenados o número serial, a chave e o valor do contador, que é incrementado a cada acionamento. Uma vez gravada, a chave é utilizada internamente no processo criptográfico, mas não pode ser lida externamente por nenhuma interface ou programa – sendo acessível apenas ao próprio circuito interno do chip durante a execução das operações criptográficas.

3.2.1. Montagem do pacote de dados

O circuito integrado HCS201 monta um pacote de dados de 66 bits, conforme ilustrado na Figura 2. Os primeiros 34 bits são transmitidos em claro e os 32 bits finais correspondem a dados cifrados com o algoritmo KeeLoq.

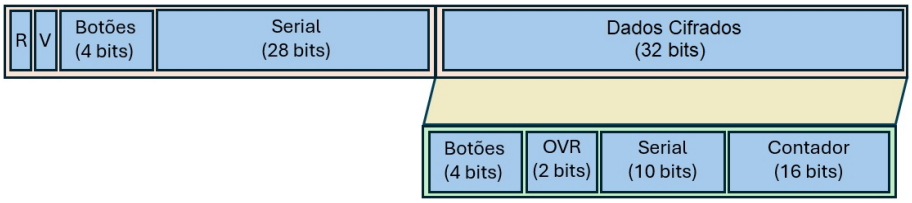


Figura 2. Pacote de dados de 66 bits montados pelo HCS201.

Os bits em claro incluem: i) R (1 bit), que indica se é a primeira transmissão do pacote (valor 0) ou uma retransmissão contínua enquanto o dispositivo permanece acionado (valor 1); ii) V (1 bit), que sinaliza baixa tensão de alimentação, útil para alertas em dispositivos com bateria; iii) *Botões* (4 bits), que indicam quais terminais do chip foram acionados, permitindo até 16 combinações distintas; e iv) *Serial* (28 bits), que representa o número serial único do chip, utilizado para identificação pelo receptor. Com 28 bits, é possível endereçar mais de 250 milhões de dispositivos.

Os 32 bits finais do pacote são cifrados e somente podem ser decifrados por quem possui a chave criptográfica correspondente. Estes incluem: i) *Bits dos botões* (4 bits), que duplicam a informação dos botões transmitida em claro; ii) *Bits* OVR (2 bits), que são incrementados quando o contador atinge seu limite e permanecem em 11 (binário) após isso; iii) *Dado serial* (10 bits), correspondentes aos 10 bits menos significativos do

número serial, usados na validação da decifração; e iv) *Contador* (16 bits), incrementado a cada acionamento para garantir a sequência e sincronização com o receptor.

A sincronização do contador é essencial para garantir a autenticidade da comunicação. Como pode ocorrer de o transmissor ser acionado sem que o receptor receba o sinal, o contador do transmissor pode avançar sem que o receptor acompanhe. Caso o receptor receba um valor com diferença de 1 a 16 em relação ao último valor válido, a sincronização é automática. Se a diferença for entre 17 e 128, são exigidas duas transmissões consecutivas válidas. Por fim, para diferenças acima de 128, o número serial é removido da lista de dispositivos confiáveis.

Essa última medida previne ataques em que um invasor tente transmitir pacotes de dados forjados com os 32 bits criptografados variando a cada transmissão, na tentativa de gerar aleatoriamente um código válido. Como a probabilidade de gerar um dos 16 valores aceitáveis ou dois valores consecutivos dentre os 112 esperados sem gerar um valor inválido é extremamente baixa, essa estratégia reforça a segurança do sistema contra esse tipo de ataque.

4. Criptanálise do circuito integrado HCS201

Nesta seção será apresentada de forma detalha a vulnerabilidade do KeeLoq e como é possível explorá-las no HCS201 usando DPA. Conforme mostrado por [Eisenbarth et al. 2008], a vulnerabilidade da implementação do algoritmo baseia-se no princípio de que o algoritmo Keeloq é implementado em hardware no HCS201 e que os circuitos de memória interna dissipam mais potencia elétrica quando há variação do estado lógico de seus bits. Como o circuito integrado em questão envia os códigos que são obtidos no final do processo criptográfico (para o acionamento do receptor), o algoritmo Keeloq pode ser atacado de trás pra frente, ou seja, começando no último ciclo executado.

A Figura 3 mostra o estado dos registradores de dados (os mesmos 32 bits de dados cifrados apresentados na Figura 2) e da chave nos três últimos ciclos do algoritmo. Os 32 bits de dados, denotados por $\mathbf{B} = (b_0, b_1, ..., b_{31})$, são conhecidos, já que são os dados presentes no registrador após todo o processo criptográfico e assim, são transmitidos pelo dispositivo. Os bits da chave, denotados por $\mathbf{K} = (k_0, k_1, ..., k_{63})$, assim como os bits denominados x_0 e x_1 na Figura 3 não são inicialmente conhecidos, pois são valores intermediários de b_0 nos ciclos 527 e 526, respectivamente. É a partir deles que será feito o ataque. Com base na descrição do KeeLoq feita na Seção 3.1.1, temos as seguintes relações:

$$\begin{aligned} b_{31} &= b_{15} \oplus NLF\left(b_{30}, b_{25}, b_{19}, b_8, b_0\right) \oplus k_{15} \oplus x_0 \\ b_{30} &= b_{14} \oplus NLF\left(b_{29}, b_{24}, b_{18}, b_7, x_0\right) \oplus k_{14} \oplus x_1 \end{aligned} \tag{1}$$

Rearranjando a Equação 1, pode-se isolar os bits x_0 e x_1:

$$\begin{aligned} x_0 &= b_{15} \oplus NLF\left(b_{30}, b_{25}, b_{19}, b_8, b_0\right) \oplus k_{15} \oplus b_{31} \\ x_1 &= b_{14} \oplus NLF\left(b_{29}, b_{24}, b_{18}, b_7, x_0\right) \oplus k_{14} \oplus b_{30} \end{aligned} \tag{2}$$

É possível perceber que o conhecimento do bit x_0 está diretamente relacionado ao bit k_{15} da chave, pois todos os outros valores relacionados são conhecidos. Sendo assim, é possível calcular o valor do bit x_0 supondo um valor para o bit k_{15} da chave. Partindo disso, é possível avaliar as duas suposições de $k_{15} \in \{0, 1\}$ e para cada uma delas classificar a amostra dos dados conhecidos em dois subconjuntos:

1. $x_0 = b_0$: o primeiro bit do registrador não sofreu mudança do estado lógico na mudança do ciclo 527 para o ciclo 528.
2. $x_0 \neq b_0$: o primeiro bit do registrador sofreu mudança do estado lógico na mudança do ciclo 527 para o ciclo 528.

A Figura 4 mostra de maneira simplificada a divisão do conjunto de todos os traços obtidos pelos acionamentos do HCS201 em quatro subconjuntos. Cada par de subconjunto é resultado das suposições ($k_{15} = 1$ e $k_{15} = 0$). Cada par se subdividem em função da igualdade ou diferença entre x_0 e b_0. Para simplificar, são apresentadas apenas 10 amostras nomeadas de A até J.

Analisando a divisão do conjunto de amostras, é possível perceber que os dois pares de subconjuntos formados por cada suposição do bit da chave ($k_{15} = 1$ e $k_{15} = 0$) são iguais mudando apenas a ordem de acordo com a igualdade ou diferença entre x_0 e b_0. Isso ocorre porque a relação do bit x_0 com o bit k_{15} da chave é linear. Neste sentido, é necessário escolher um valor que tenha um alto grau de não linearidade em relação à chave [Eisenbarth et al. 2008].

Como as duas suposições para o bit k_{15} da chave são simétricas, torna-se impossível avaliar qual é a suposição correta. Já o conhecimento do bit x_1 está diretamente relacionado ao bit k_{14} da chave e ao bit x_0. Como o bit x_0 está relacionado ao bit k_{15} da chave, o bit x_1 também está relacionado com o bit k_{15} da chave como dado de entrada da NLF. Isso pode ser demonstrado rearranjando a Equação 2:

$$x_1 = b_{14} \oplus NLF\left(b_{29}, b_{24}, b_{18}, b_7, (b_{15} \oplus NLF\left(b_{30}, b_{25}, b_{19}, b_8, b_0\right) \oplus k_{15} \oplus b_{31})\right) \oplus k_{14} \oplus b_{30} \quad (3)$$

Ainda, o conhecimento do bit x_1 está diretamente relacionado aos bits k_{15} e k_{14} da chave, pois todos os outros valores relacionados são conhecidos. Sendo assim, é possível

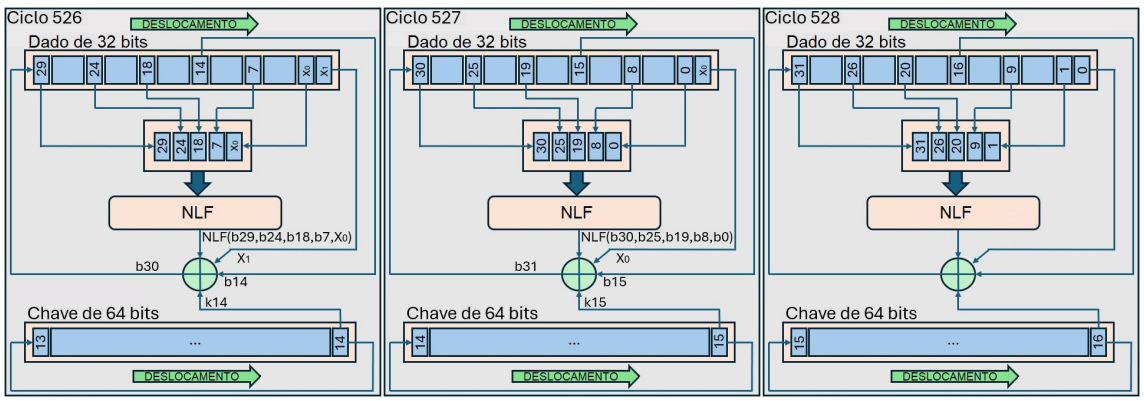


Figura 3. Estado dos registradores.

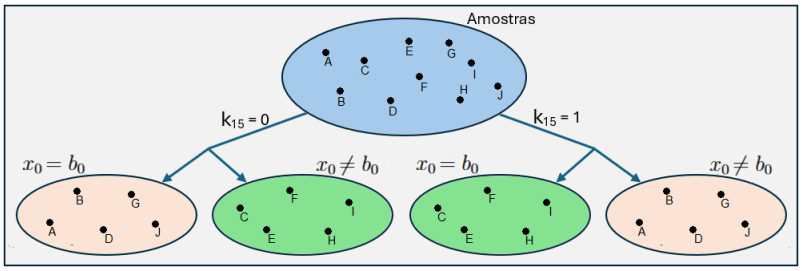


Figura 4. Divisão das amostras em função de x_0 e b_0.

calcular o valor do bit x_1 supondo um valor para o bit k_{15} da chave e um valor para o bit k_{14} da chave. No entanto, como o bit k_{14} da chave está relacionado ao bit x_1 de forma linear, define-se qualquer valor para ele e avalia-se as duas suposições de $k_{15} \in \{0, 1\}$ e para cada uma delas classifica-se a amostra dos dados conhecidos em dois tipos:

1. $x_1 = x_0$: o primeiro bit do registrador não sofreu mudança do estado lógico na mudança do ciclo 526 para o ciclo 527.
2. $x_1 \neq x_0$: o primeiro bit do registrador sofreu mudança do estado lógico na mudança do ciclo 526 para o ciclo 527.

A Figura 5 mostra de maneira simplificada a divisão do conjunto de todos os traços obtidos pelos acionamentos do HCS201 em quatro subconjuntos. Cada par de subconjunto é resultado das suposições ($k_{15} = 1$ e $k_{15} = 0$). Cada par se subdividem em função da igualdade ou diferença entre x_1 e x_0. Para simplificar, são apresentadas apenas 10 amostras nomeadas de A até J. Além dos dados $\mathbf{B} = (b_0, b_1, ..., b_{31})$, cada uma das amostras contem o traço de potência emitido pelo HCS201 durante o seu respectivo acionamento.

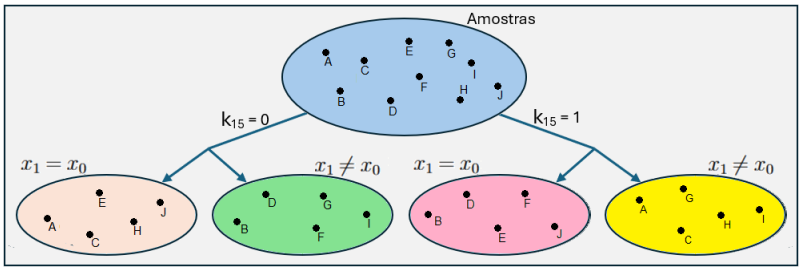


Figura 5. Divisão das amostras em função de x_1 e x_0.

Analisando a divisão do conjunto de amostras é possível perceber que os dois pares de subconjuntos gerados são formados de forma totalmente diferente para cada suposição de valor do bit k_{15} da chave. Isso ocorre porque a relação do bit x_1 com o bit k_{15} da chave não é linear pois o bit da chave k_{15} se relaciona com o bit x_1 como um dado de entrada da função NLF. Como o bit k_{14} da chave está relacionado com o bit x_1 de forma linear, a mudança da suposição de seu valor apenas irá inverter a formação dos pares de subconjuntos, tornando a suposição de seu valor não relevante.

Como os dois pares de subconjunto gerados são totalmente diferentes para cada suposição do valor do bit k_{15} da chave, através de DPA é possível explorar a correlação entre o consumo de potência e a manipulação de bits internos do dispositivo, já que apenas

a suposição correta do bit k_{15} da chave apresentará dois subconjuntos de amostras de dados de saída cujos consumos de potência se diferem significativamente. Isso ocorre porque apenas para a suposição correta o subconjunto $x_0 \neq x_1$ será formado por amostras cujo consumo de potência será mais alto durante os 31 últimos ciclos em que estes dados deslocam pelo barramento de dados já que nestes 31 ciclos o bit de memória que contém x_0 passará a conter x_1, ou seja, uma variação de estado lógico.

Embora os outros bits de dados também apresentem variações de estado lógico e, consequentemente, consumo de potência, mascarando o consumo de potência que revela o bit k_{15} da chave, estatisticamente, se for usada uma quantidade de amostras suficientemente grande, estas variações de potência serão distribuídas por igual nos dois subconjuntos. Sendo assim, basta calcular a média do consumo de potência referente aos dois subconjuntos separadamente e compará-los. Os dados referentes ao consumo de potência de outros bits e até mesmo de eventuais ruídos tenderão a se anularem sobrando apenas o consumo de potência referente ao bit k_{15} da chave. Como a diferença do consumo de potência só existirá se os dois subconjuntos tiverem de fato valores de consumo de potência distintos, analisando os traços da diferença dos consumos de potência para as duas suposições de valores para o bit k_{15} da chave, apenas um dos traços irá apresentar um pico positivo ou negativo que se destacará do valor médio de todo o traço e assim revela-se o valor correto do bit k_{15} da chave

Uma vez conhecido o bit k_{15} da chave, pode-se conhecer o valor do bit x_0 e todos os estados lógicos dos registradores durante o ciclo 527. Sendo assim, basta repetir a análise, só que ao invés de analisar os ciclos 526 e 527, analisa-se os ciclos 525 e 526, pois assim é possível obter o bit k_{14} da chave. Com o bit k_{14} da chave revelado, executa-se o processo de análise dos dados nos ciclos anteriores e obtém-se o bit k_{13}, que por sua vez possibilita repetir o processo para obter o bit k_{12} e assim sucessivamente até o último bit da chave ser revelado. O processo de análise dos dados precisa ser repetido para cada bit da chave. No entanto, isso não se apresenta como um problema uma vez que essa análise é rápida e fácil de ser processada.

5. Implementação do ataque

Nesta seção será apresentado todo o processo realizado para a quebra da chave criptográfica do circuito integrado HCS201 descrevendo as fases de coleta de dados e do processo de DPA para a recuperação da chave secreta.

Para realizar tais medições, [Eisenbarth et al. 2008] utilizaram um osciloscópio Agilent 54832D, com taxa de amostragem de até 4 GS/s e capacidade de armazenamento de 128 Mpts, o que possibilita a captura de sinais de alta frequência com elevada resolução temporal e durante janelas de tempo relativamente longas. Graças a essa taxa de amostragem elevada, os dados obtidos apresentaram excelente fidelidade, permitindo a extração da chave criptográfica com apenas algumas dezenas de traços. Essa alta precisão reduziu significativamente a necessidade de coleta e processamento de grandes volumes de dados.

Em contraste, nossa proposta consistiu na construção de um circuito dedicado para aquisição de sinais, com custo extremamente reduzido. Esse circuito foi capaz de operar com uma taxa de amostragem de apenas 250 kS/s, muito inferior àquela oferecida por osciloscópios profissionais. Embora tenha sido possível armazenar um número razoável de amostras, a baixa resolução temporal e a ausência de um sistema de trigger preciso

tornaram os dados coletados mais ruidosos e imprecisos. Como consequência, foi necessário coletar aproximadamente 10 mil traços, além de implementar etapas adicionais de processamento, como alinhamento por picos de consumo e filtragem de ruído, a fim de viabilizar a extração da chave criptográfica. A Tabela 5 apresenta uma comparação entre as características do trabalho de Eisenbarth *et al.* e as características deste trabalho.

Tabela 1. Comparação entre [Eisenbarth et al. 2008] e este trabalho.

	[Eisenbarth et al. 2008]	Este trabalho
Hardware de aquisição	Agilent 54832D	Circuito próprio
Taxa de amostragem	Até 4 GS/s	250 kS/s
Precisão das amostras	Alta	Baixa
Volume necessário de dados	Baixo	Alto
Capacidade de armazenamento	128 Mpts	500 kpts
Custo	Acima de US$ 1.000,00	Abaixo de US$ 25,00

Embora a técnica de exploração do bit x_0, baseada na Equação 2, derive de observações originalmente apresentadas por Eisenbarth *et al.*, o modelo estatístico empregado em nosso trabalho foi significativamente simplificado para operar com hipóteses binárias diretas (mudança ou não de estado lógico). Isso permitiu uma análise mais compatível com o uso de hardware acessível e limitado. Assim, não utilizamos o CPA tradicional, que exige a definição de modelos de vazamento (como Hamming Weight ou Hamming Distance) e a extração precisa de picos. Em vez disso, agrupamos os traços com base na previsão de transição de bits, estratégia mais robusta frente ao ruído e à baixa resolução dos traços.

Dessa forma, ainda que inspirado na abordagem anterior, este trabalho é inédito ao demonstrar que esse tipo de ataque pode ser realizado com sucesso utilizando apenas equipamentos de baixo custo, o que amplia significativamente seu impacto prático. Ao eliminar a barreira econômica associada à instrumentação sofisticada, a ameaça ao KeeLoq se torna ainda mais crítica, viabilizando ataques reais em cenários antes considerados inviáveis, como discutido na Seção 4.

5.1. Descrição do hardware de baixo custo

Para este trabalho, o STM32H743ZI foi escolhido devido à alta frequência operacional de sua CPU (500 MHz), à disponibilidade suficiente de memória RAM (564 kB) para armazenar as amostras coletadas, e à presença de um conversor analógico-digital (ADC) integrado de 8 bits, além do seu baixo valor de mercado, abaixo de US$ 20,00. Com essa configuração, é possível obter uma taxa de amostragem de aproximadamente 250 kS/s, o que equivale a 4 μs entre amostras. Portanto, a taxa de amostragem do ADC do STM32H743ZI é relativamente baixa, algo comum em dispositivos de baixo custo, especialmente se comparada as taxas obtidas por equipamentos dedicados de medição, como osciloscópios.

A Figura 6 apresenta o hardware de baixo custo desenvolvido. O circuito foi construído de forma modular, permitindo que outros chips, não somente aqueles da família

HCS, também possam ser analisados futuramente. A placa à esquerda abriga o chip HCS201, alvo da extração da chave criptográfica, além de alguns *jumpers* utilizados para selecionar qual porta será acionada durante a captura dos dados. A placa central contém o circuito integrado OPA4354A, um amplificador operacional de alta velocidade, responsável por amplificar os sinais de consumo de corrente do dispositivo. Já a placa à direita contém o módulo STM32H743ZI, anteriormente mencionado, que realiza a amostragem e transmissão dos dados. Esses módulos estão interligados por um cabo *flat*, que conecta dois periféricos adicionais: um conversor USB/Serial e um gravador ST-LINK V2. O conversor USB/Serial estabelece a comunicação entre o microcontrolador e o computador responsável pela coleta e armazenamento dos dados. O gravador ST-LINK V2, por sua vez, é utilizado tanto para carregar o *firmware* no microcontrolador quanto para fornecer alimentação elétrica ao circuito durante sua operação.

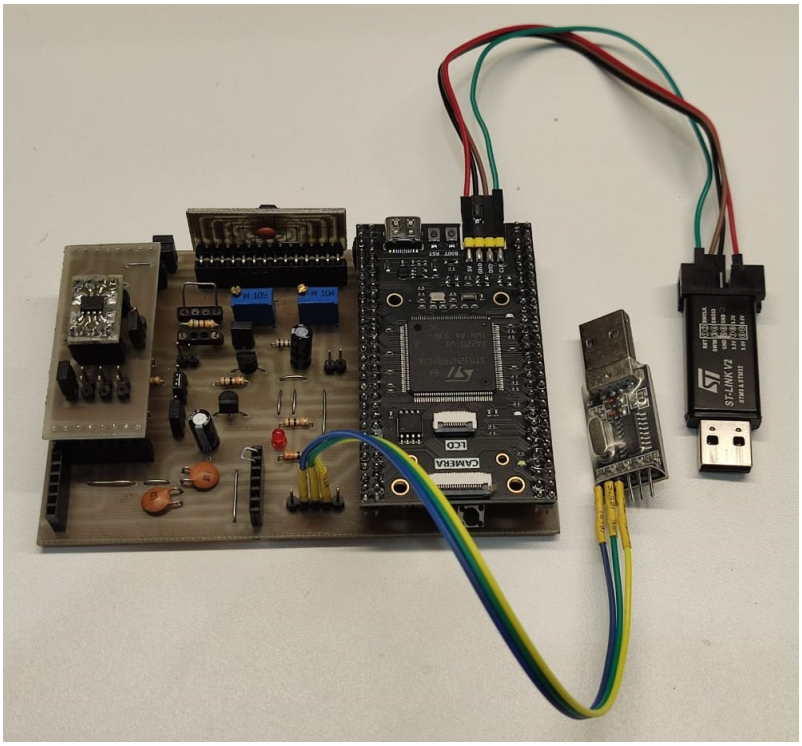


Figura 6. Hardware de baixo custo construido com PCB.

5.2. Coleta de dados

Para que o DPA seja aplicado, é necessário medir o consumo de potência do circuito integrado HCS201 com precisão. Para isso, foi utilizado um resistor conectado ao terminal negativo de alimentação do dispositivo, garantindo que toda a corrente consumida pelo HCS201 passe pelo resistor. A corrente consumida gera uma pequena queda de tensão no resistor, que é proporcional ao consumo de potência e pode ser amplificada e medida diretamente. A captura dessa diferença de potencial foi feita utilizando um amplificador operacional de baixo ruído, elevando o sinal para um nível adequado ao ADC integrado no STM32H743ZI.

Para compensar a limitação da taxa de amostragem do STM32H743ZI, foi necessário coletar um volume maior de dados, permitindo minimizar a imprecisão das

medições individuais através da média estatística das múltiplas leituras realizadas. Portanto, para a coleta de dados, a cada 2 segundos o próprio microcontrolador aciona o HCS201, faz a coleta de 6 mil amostras do consumo de potência, o que equivale a um intervalo de 24 ms, e armazena em sua memória RAM interna. Em seguida, efetua a leitura do sinal elétrico emitido pelo HCS201 para obter os dados calculados após o processo criptográfico (pacote de dados de 66 bits ilustrado na Figura 2). Por fim, envia os dados para um computador através de sua porta serial. No total, foram realizados 10 mil acionamentos do HCS201, cada acionamento constituído por 6 mil amostras do consumo de potência. Sendo assim, a fase de coleta levou menos de 6 horas.

5.3. Análise diferencial de potência

A Figura 7 apresenta o gráfico do consumo de potência do circuito integrado HCS201 durante um acionamento, abrangendo um conjunto de 6 mil amostras adquiridas pelo ADC.

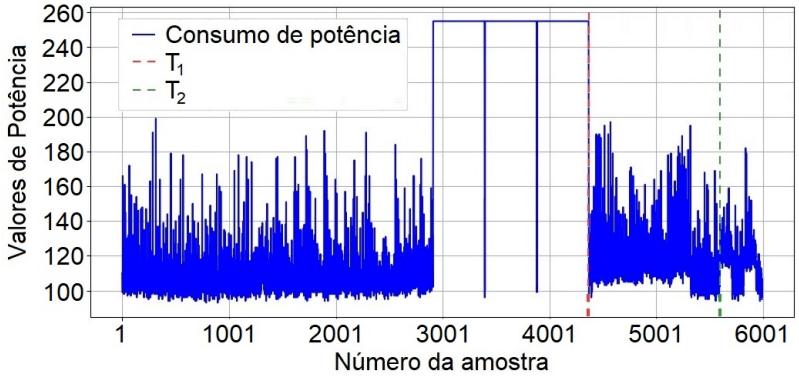


Figura 7. Consumo de potência de um acionamento do HCS201.

Observa-se que o consumo de potência medido varia entre os valores digitais aproximados de 95 e 200. Nota-se claramente, no entanto, que no meio do intervalo analisado há um aumento expressivo no consumo, chegando a saturar a medição do ADC. Esse alto consumo de potência ocorre durante o processo de leitura e escrita na EEPROM interna do HCS201 [Eisenbarth et al. 2008]. Dado que o processo criptográfico deve ocorrer após o acesso à EEPROM, tendo que precisa-se acessar o número serial, a chave e o número de contador armazenado nela, pode-se supor que a operação de criptografia ocorre imediatamente após essa elevação abrupta do consumo de potência, identificado no gráfico como T_1. Além disso, no ponto identificado no gráfico como T_2, observa-se outra variação perceptível no consumo, que coincide com a geração do sinal de saída no pino correspondente do circuito integrado. Esta região final representa, portanto, o consumo decorrente da transmissão efetiva do sinal criptografado (pacote de dados). Tal período encontra-se claramente diferenciado e ocorre após a região de criptografia mencionada anteriormente. Essas observações permitem concluir que o trecho do gráfico mais adequado para análise via DPA está situado logo após o pico de consumo de potência (T_1) e antes do início da transmissão do sinal criptografado (T_2).

A Figura 8 apresenta o gráfico do consumo de potência de dois acionamentos distintos do circuito integrado HCS201, abrangendo o período que se inicia pouco antes do término do acesso à memória EEPROM e se estende até logo após o início da transmissão

do sinal de saída. Nota-se que os traços de consumo apresentam certa dessincronização, o que é esperado, uma vez que o HCS201 não possui um oscilador interno de alta precisão para garantir tempos de execução idênticos entre acionamentos consecutivos. Para viabilizar a aplicação da DPA e a extração da chave criptográfica, é essencial sincronizar todas as 10 mil amostras de dados coletadas. Esse alinhamento é realizado tomando como referência dois pontos característicos no gráfico: a queda brusca no consumo de potência logo após o acesso à EEPROM (T_1) e o aumento significativo de consumo de potência no início da transmissão do sinal de saída (T_2). Esses momentos foram escolhidos por serem facilmente identificáveis, permitindo um realinhamento preciso dos traços e garantindo a eficácia da análise.

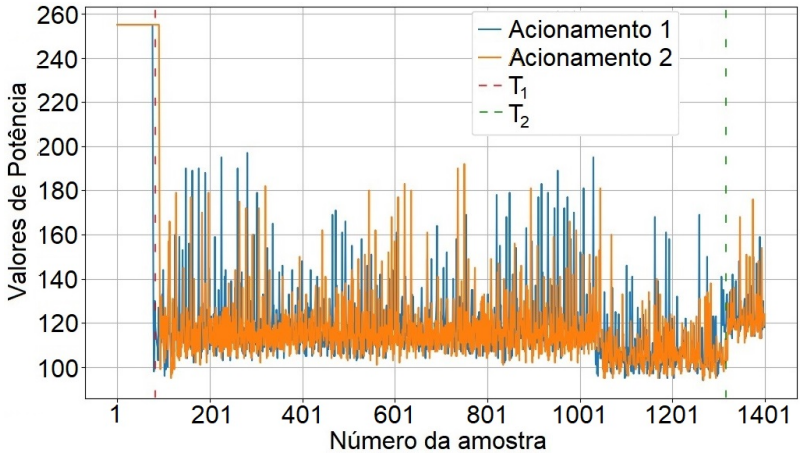


Figura 8. Consumo de potência de dois acionamentos do HCS201.

Para a sincronização, todas as amostras anteriores a T_1 e posteriores a T_2 foram removidas, assegurando que os traços dos 10 mil acionamentos iniciem exatamente em T_1 e terminem em T_2. No entanto, alguns traços possuem um número maior de amostras que outros devido à imprecisões do HCS201. Para padronizar a quantidade de amostras em todos os traços, foram inseridas amostras adicionais nos traços com menor quantidade de pontos, preservando as características temporais e a coerência dos dados. O processo de ajuste ocorre em etapas. Primeiramente, determina-se o traço com o maior número de amostras, que será utilizado como referência. Em seguida, cada traço com menor quantidade de amostras é ajustado para igualar sua contagem à do traço de referência. O número total de amostras a serem adicionadas é calculado e distribuído uniformemente ao longo do traço. Por exemplo, se for necessário inserir duas amostras adicionais, o traço é dividido em três segmentos de tamanho aproximadamente igual. Nos pontos de inserção, calcula-se a média aritmética entre os valores digitalizados das amostras adjacentes, garantindo que as novas amostras sejam inseridas de forma suave e consistente, minimizando distorções no sinal original. Além disso, passou-se um filtro passa-baixa média móvel em cada um dos traços de potência, removendo componentes de alta frequência que não contribuem para a análise e atrapalham na detecção. Após a aplicação do filtro passa-baixa, aplica-se a DPA baseando-se na vulnerabilidade do KeeLoq discutido na Seção 4.

Inicialmente, supõe-se que o bit k_{15} da chave é 0 e então dividi-se todos os traços em dois grupos. O primeiro grupo corresponde aos traços em que não há variação de

estado lógico no primeiro bit do barramento de dados no deslocamento do ciclo 526 para o ciclo 527 durante o algoritmo criptográfico. Já o segundo grupo corresponde aos traços em que há variação. Uma vez definidos os dois grupos, calcula-se a média de todos os traços de cada grupo, obtendo dois traços médios. Subtraindo-se um traço médio pelo outro obtêm-se um traço diferencial. Em seguida, repete-se o processo, mas supondo que o bit k_{15} da chave é 1. Novamente, dividi-se todos os traços em dois grupos, gera-se dois traços médios e finalmente um novo traço diferencial. Dessa forma tem-se dois traços diferenciais, sendo um o resultado da suposição que o bit k_{15} da chave é 0 e outro de que o bit k_{15} da chave é 1. Os dois traços diferenciais tendem a ter seus valores próximos do valor zero. No entanto, o traço diferencial da suposição correta deverá ter um pico muito evidente, pico este que representa a potência consumida no momento em que os dois bits de dados analisados deslocam pelo barramento de dados durante o processo criptográfico.

A Figura 9 mostra os dois traços diferenciais obtidos por DPA para a suposição de que $k_{15} = 0$ e $k_{15} = 1$. É possível perceber que a suposição de que $k_{15} = 1$ gerou um pico evidente, indicando que essa é a suposição correta.

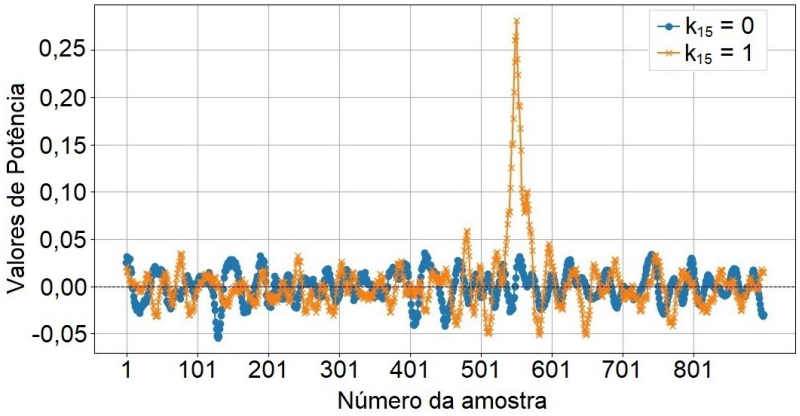


Figura 9. Diferença dos traços médios.

Uma vez que o bit k_{15} da chave é conhecido, a DPA pode ser repetido para descobrir o bit k_{14} da chave, pois o conhecimento do bit k_{15} da chave permite conhecer o estado do barramento de dados no ciclo 527, conforme discutido na Seção 4. Os traços podem ser novamente divididos em dois grupos baseados na suposição do valor do bit k_{14} da chave gerando-se quatro traços médios e, por fim, dois traços diferenciais. Estes irão revelar o correto valor do bit k_{14} da chave. Esse processo é então repetido para os demais bits da chave até que se obtenha a chave completa. Devido à este processo ser puramente computacional e não exigir interface com o HCS201, apenas com os dados já coletados, esta etapa é processada de forma muito rápida, na ordem de minutos.

5.4. Aplicando o ataque

A coleta dos dados e DPA, descritas nas Seções 5.2 e 5.3, foi realizada em dois circuitos integrados HCS201 de dois controles remotos de portão eletrônicos de duas empresas distintas. Após a execução do DPA, foram obtidas as chaves criptográficas supostamente utilizadas em cada um dos controles remotos analisados. A validação do sucesso do ataque foi feita utilizando a chave extraída para decifrar algumas das 10 mil transmissões

criptografadas capturadas durante os acionamentos do HCS201. Para isso, foi implementada a função de decifração do KeeLoq em Python e verificou-se que, após a decifração, os valores do contador presentes nos dados transmitidos seguiam uma sequência coerente, conforme esperado. Além disso, a extração bem-sucedida da chave foi confirmada ao comparar os dez primeiros bits do número serial decifrado com o número serial transmitido em claro, demonstrando que a chave recuperada corresponde exatamente à utilizada pelo chip para cifrar os dados.

Após a confirmação da chave extraída, foram analisados dados de outros controles remotos das mesmas empresas utilizando um receptor de radiofrequência, com objetivo de verificar se a mesma chave criptográfica era empregada em diferentes dispositivos de um mesmo fabricante. Os resultados mostraram que ambas as empresas analisadas utilizam a mesma chave criptográfica em todos os seus controles remotos, sem a aplicação de qualquer método de derivação de chave para proteção adicional do sistema criptográfico. Essa prática representa uma vulnerabilidade crítica, pois a extração bem-sucedida da chave de um único controle remoto compromete todos os dispositivos da mesma marca, permitindo ataques de clonagem e pirataria em larga escala. Dessa forma, qualquer adversário capaz de recuperar a chave de um controle específico pode replicá-la para outros dispositivos do mesmo fabricante, comprometendo toda a segurança do sistema.

Uma vez que se conheça a chave criptográfica, um atacante é capaz de capturar o sinal de um transmissor e, a partir dele, calcular os próximos sinais que o transmissor irá transmitir quando for novamente acionado. Dessa forma, pode-se acessar o receptor transmitindo o mesmo sinal que o transmissor irá transmitir futuramente quando for novamente acionado e até mesmo retirar o acesso do transmissor cujo sinal foi capturado bastando para isso calcular e transmitir o valor do contador com 128 incrementos, conforme discutido na Seção 3.2.1, apagando o transmissor de sua lista de acesso.

5.5. Mitigação

Para mitigar os efeitos do ataque de DPA apresentado neste trabalho em um sistema de segurança baseado no KeeLoq, deve-se utilizar um processo de derivação de chaves baseado, por exemplo, em função de derivação de chaves baseada em HMAC, do inglês HMAC-based key derivation function (HKDF). Este processo consiste em gerar uma chave única K_D, chamada de chave do dispositivo, para cada transmissor (controle remoto) a partir da chave mestra K_M e do serial de cada transmissor. A chave K_M ainda deve ser gravada no receptor, permitindo assim o acionamento do portão eletrônico pelo transmissor. Note que é necessário que o receptor tenha conhecimento do HKDF usado para que o mesmo seja capaz de obter a chave K_D com base no serial recebido no pacote. Neste cenário, partindo do princípio que o HKDF é uma *one-way function* segura e que K_M seja apenas conhecida pela empresa, uma extração de K_D não revela nada sobre K_M. Portanto, um atacante que aplica um ataque DPA bem sucedido em um controle remoto apenas será capaz de acionar um receptor, aquele que é seu par, mas não será capaz de atacar outros pares de transmissor/receptor, exceto no caso em que também faça a quebra deste outro transmissor. É importante notar que essa mitigação não se aplica aos dispositivos já comercializados, tendo em vista que é inviável realizar a regravação de todos eles. Sendo assim, os sistemas de segurança que utilizam os chips HCS201 sem qualquer derivação de chaves ficarão expostos mesmo que apenas um único transmissor tenha sua chave extraída, tendo em vista que, neste caso, esta chave funciona como uma chave

mestra.

Uma outra forma de mitigação possível é reprojetar o HCS201 para que o mesmo não seja vulnerável à ataques por DPA. Para isso, o hardware deve ser remodelado para que seu consumo de potência seja constante ou que seu consumo não revele nada sobre a chave criptográfica, por exemplo, usando mascaramento. No entanto, assim como a mitigação anterior, esta mitigação necessitaria da troca de todos os equipamentos já comercializados, não sendo uma solução viável. Porém, tal medida poderia proteger futuros sistemas de segurança que ainda serão comercializados. Observa-se então que neste caso, a combinação das mitigações propostas é um caminho conservador para garantir a segurança de sistemas baseados no KeeLoq.

6. Conclusão

Este trabalho demonstrou um ataque de baixo custo ao circuito integrado HCS201, utilizado em sistemas de controle de acesso baseados no KeeLoq. Através de DPA, foi possível extrair a chave criptográfica do dispositivo, confirmando a vulnerabilidade da implementação. Diferentemente de abordagens anteriores, que dependem de equipamentos de alto custo, a metodologia proposta emprega hardware acessível, viabilizando a realização do ataque com recursos limitados. Os experimentos ainda mostraram que os fabricantes analisados utilizam a mesma chave criptográfica em todos os seus dispositivos, tornando-os suscetíveis a clonagem e pirataria em larga escala. Como mitigação, foi sugerida a derivação de chaves baseada em HKDF, garantindo chaves únicas para cada transmissor e impedindo que a extração de uma única chave comprometa todo o sistema. Os resultados ressaltam a importância de considerar ataques por canal lateral no desenvolvimento de dispositivos embarcados. Como trabalhos futuros, pretende-se avaliar contramedidas de software que possam mitigar esse tipo de ataque sem necessidade de modificações no hardware.

Referências

[Bogdanov 2007] Bogdanov, A. (2007). Attacks on the keeloq block cipher and authentication systems. In *3rd Conference on RFID Security*, volume 2007.

[Courtois et al. 2008] Courtois, N. T., Bard, G. V., and Wagner, D. (2008). Algebraic and slide attacks on keeloq. In *Fast Software Encryption: 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers 15*, pages 97–115. Springer.

[Eisenbarth et al. 2008] Eisenbarth, T., Kasper, T., Moradi, A., Paar, C., Salmasizadeh, M., and Shalmani, M. T. M. (2008). On the power of power analysis in the real world: A complete break of the keeloq code hopping scheme. In *Advances in Cryptology–CRYPTO 2008: 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings 28*, pages 203–220. Springer.

[Indesteege et al. 2008] Indesteege, S., Keller, N., Dunkelman, O., Biham, E., and Preneel, B. (2008). A practical attack on keeloq. In *Advances in Cryptology–EUROCRYPT 2008: 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings 27*, pages 1–18. Springer.

[Ji et al. 2023] Ji, Y., Wang, R., Ngo, K., Dubrova, E., and Backlund, L. (2023). A side-channel attack on a hardware implementation of crystals-kyber. In *2023 IEEE European Test Symposium (ETS)*, pages 1–5. IEEE.

[Kocher et al. 1999] Kocher, P., Jaffe, J., and Jun, B. (1999). Differential power analysis. In *Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*, pages 388–397. Springer.

[Kocher 1996] Kocher, P. C. (1996). Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology—CRYPTO'96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings 16*, pages 104–113. Springer.

[Marneweck 1996] Marneweck, K. (1996). *An introduction to keeloq code hopping*. Microchip Technology Inc. Rev. 1.0.

[Microchip 2001] Microchip, T. I. (2001). *HCS201 - KEELOQ® Code Hopping Encoder*. Microchip Technology Inc. Rev. 1.0.

[Paar et al. 2009] Paar, C., Eisenbarth, T., Kasper, M., Kasper, T., and Moradi, A. (2009). Keeloq and side-channel analysis-evolution of an attack. In *2009 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 65–69. IEEE.

[Soto-Cruz et al. 2024] Soto-Cruz, J., Ruiz-Ibarra, E., Vázquez-Castillo, J., Espinoza-Ruiz, A., Castillo-Atoche, A., and Mass-Sanchez, J. (2024). A survey of efficient lightweight cryptography for power-constrained microcontrollers. *Technologies*, 13(1):3.

[Sun et al. 2025] Sun, P., Wan, Y., Wu, Z., Fang, Z., and Li, Q. (2025). A survey on privacy and security issues in iot-based environments: Technologies, protection measures and future directions. *Computers & Security*, 148:104097.

[Thabit et al. 2023] Thabit, F., Can, O., Aljahdali, A. O., Al-Gaphari, G. H., and Alkhzaimi, H. A. (2023). Cryptography algorithms for enhancing iot security. *Internet of Things*, 22:100759.

FrodoKEM based on multicore implementation: enhancing performance and memory efficiency in IoT applications

Marcos V. C. Amorim[1], Vinícius Lagrota[2], Mateus de L. Filomeno[1], Moisés V. Ribeiro[1]

[1] Electric Engineering Department
Federal University of Juiz de Fora (UFJF) – Juiz de Fora, MG – Brazil

[2]Research and Development Center for Communication Security (CEPESC)
Brasília, DF – Brazil

marcos.amorim@estudante.ufjf.br, vinicius.lagrota@gmail.com,
delimafilomeno@gmail.com, moises.ribeiro@ufjf.br

Abstract. *This paper presents a multicore implementation of FrodoKEM, a post-quantum cryptographic scheme, targeting near real-time Internet of Things (IoT) applications. By combining tiled matrix multiplication and data parallelism, we enable FrodoKEM to efficiently operate under the memory and performance constraints of multicore microcontrollers. Implemented on the ESP32-S3, the single-core version demonstrates feasibility through memory optimization, while the dual-core version achieves a* 44.5% *reduction in execution time, meeting the latency requirements of IoT systems.*

1. Introduction

Cryptography is essential for securing digital communications, playing a key role in protecting sensitive data across various applications, including financial transactions, e-commerce, and secure messaging. In IoT systems, where interconnected devices exchange critical data, cryptographic protocols ensure data confidentiality and integrity. For example, in smart grids, secure communication between utility providers and smart meters is vital [Costa et al. 2022].

The advent of cryptographically relevant quantum computers (CRQCs) threatens conventional cryptographic schemes such as Rivest-Shamir-Adleman (RSA) and elliptic curve cryptography (ECC), which are vulnerable to quantum algorithms like Shor's [Shor 1994]. In response, post-quantum cryptography (PQC) has emerged as a solution based on quantum-resistant mathematical problems. Since 2017, National Institute for Standards and Technology (NIST) has led efforts to standardize PQC, culminating in the selection of Module-Lattice-Based Key Encapsulation Mechanism (ML-KEM) as the primary Key Encapsulation Mechanism (KEM) for general-purpose use. Nevertheless, certain scenarios—such as critical infrastructure—demand conservative designs prioritizing long-term confidentiality.

FrodoKEM [Alkim et al. 2020], grounded in the well-established Learning With Errors (LWE) problem, was recommended by the Federal Office for Information Security (BSI) for such high-assurance applications [BSI 2020]. However, FrodoKEM's resource-intensive nature—especially in terms of memory and execution time—challenges its deployment in embedded systems typical of IoT ecosystems.

Several studies have proposed optimizations to mitigate these challenges. These include GPU-based Advanced Encryption Standard (AES) acceleration [Lee et al. 2022], matrix multiplication optimizations [Bos et al. 2021], hardware/software co-designs on RISC-V and SoC platforms [Karl et al. 2022, Costa et al. 2022], vectorized implementations using ARMv8 NEON [Kwon et al. 2021], and memory-efficient embedded versions [Bos et al. 2023]. Yet, multicore implementations of FrodoKEM remain underexplored.

This work proposes a multicore approach combining tiled matrix multiplication and data parallelism to enhance FrodoKEM's feasibility and performance in constrained microcontrollers. Implementations on the ESP32-S3 demonstrate that tiled multiplication enables FrodoKEM execution within memory limits, while data parallelism achieves a 44.5% reduction in execution time. The proposed method supports near real-time execution and complies with FrodoKEM specifications, providing a viable path for deploying quantum-resistant security in IoT environments.

The remainder of this paper is organized as follows: Section 2 outlines FrodoKEM operations. Section 3 presents the proposed multicore implementation. Section 4 details the practical setup. Section 5 discusses results and conclusions.

2. The FrodoKEM scheme

FrodoKEM [Alkim et al. 2020] is a lattice-based cryptographic scheme built upon the hardness of the LWE problem [Regev 2010]. We adopt FrodoKEM's original notation [Alkim et al. 2020]: matrices are denoted by uppercase bold letters, vectors by lowercase bold letters, $\mathbb{Z}_q$ represents integers modulo q, and $||$ indicates concatenation.

FrodoKEM emphasizes conservative security by relying on simple operations (e.g., addition, multiplication) and omitting structured lattices. It supports three parameter sets: FrodoKEM-640, -976, and -1344, targeting security levels equivalent to AES-128, -192, and -256, respectively. However, its large matrix sizes—summarized in Table 1—pose significant challenges for embedded implementations.

Table 1. Public key, private key, and ciphertext sizes for FrodoKEM (in bytes).

	Public key size	**Private key size**	**Ciphertext size**
FrodoKEM-640	9616	19888	9720
FrodoKEM-976	15632	31296	15744
FrodoKEM-1344	21520	43088	21632

The scheme consists of three main operations: key generation, encapsulation, and decapsulation. In key generation, the main computational task is forming matrix $\mathbf{B} \in \mathbb{Z}_q^{n \times \overline{n}}$ as

$$\mathbf{B} \leftarrow \underbrace{\mathbf{AS}}_{\mathbf{D}} + \mathbf{E}, \tag{1}$$

where $\mathbf{A}$ is deterministically generated from a pseudorandom seed, and $\mathbf{S}$, $\mathbf{E}$ are sampled from a discrete Gaussian distribution χ.

In encapsulation, additional noise matrices $\mathbf{S}'$, $\mathbf{E}'$, and $\mathbf{E}''$ are sampled. The following core operations are performed:

$$\mathbf{B}' \leftarrow \mathbf{S}'\mathbf{A} + \mathbf{E}', \quad \mathbf{V} \leftarrow \mathbf{S}'\mathbf{B} + \mathbf{E}''. \tag{2}$$

A message μ is encoded into matrix $\mathbf{C}$ using $\mathbf{V}$, forming part of the ciphertext alongside $\mathbf{B}'$.

During decapsulation, the ciphertext is unpacked and used to recover μ' via:

$$\mathbf{M} \leftarrow \mathbf{C} + \mathbf{B}'\mathbf{S}. \tag{3}$$

A re-encapsulation process computes:

$$\mathbf{B}'' \leftarrow \mathbf{S}'\mathbf{A} + \mathbf{E}', \quad \mathbf{C}' \leftarrow \text{Encode}(\mu'). \tag{4}$$

If $\mathbf{B}'' = \mathbf{B}'$ and $\mathbf{C}' = \mathbf{C}$, the shared secret is derived deterministically; otherwise, a random value is returned.

FrodoKEM's performance bottlenecks arise primarily from matrix multiplications involving $\mathbf{A}$, which motivate the optimizations explored in this work.

3. Proposal of multicore implementation for FrodoKEM

This section presents a multicore implementation strategy for FrodoKEM, targeting its performance bottlenecks. According to [Costa et al. 2022], operations involving matrix $\mathbf{A}$ dominate runtime when using Secure Hash Algorithm and Keccak (SHAKE)-128, accounting for over 90% of total execution time. Specifically, $\mathbf{B} \leftarrow \mathbf{AS} + \mathbf{E}$ consumes 8.22%, while $\mathbf{B}' \leftarrow \mathbf{S}'\mathbf{A} + \mathbf{E}'$ and $\mathbf{B}'' \leftarrow \mathbf{S}'\mathbf{A} + \mathbf{E}'$ together take 30.94%. The standalone generation of $\mathbf{A}$ via SHAKE-128 accounts for 53.90%.

To address these costs, we propose a combined approach using data parallelism and tiled matrix multiplication. Data parallelism distributes matrix operations across U cores, enabling concurrent processing and improved performance. However, FrodoKEM also imposes high memory demands due to large matrix dimensions—especially for matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$.

To overcome this, we apply tiled multiplication, which divides $\mathbf{A}$ into smaller row-wise blocks (tiles) that can fit into limited internal memory. Each tile is generated on-the-fly using SHAKE-128, multiplied by $\mathbf{S}$, and accumulated into the resulting matrix $\mathbf{B}$ alongside $\mathbf{E}$. The next tile then overwrites the previous one, avoiding the need to store the entire matrix $\mathbf{A}$ in memory. This technique significantly reduces the memory footprint without altering the FrodoKEM specification, making the scheme viable for resource-constrained microcontrollers.

Subsection 3.1 describes the multicore and tiled implementation of $\mathbf{AS}$, while Subsection 3.2 details the strategy for computing $\mathbf{S}'\mathbf{A}$[1].

3.1. A by S matrix multiplication

To compute the matrix multiplication of $\mathbf{A}$ by $\mathbf{S}$, which results in the matrix $\mathbf{D} \in \mathbb{Z}_q^{n \times \overline{n}}$ in (1), the data parallelism of the matrix $\mathbf{A}$ in U cores is proposed. This procedure is

[1]The implementation source code is available in `https://github.com/mv-amorim/frodokem`.

illustrated in Figure 1. The main strategy is to divide the matrix $\mathbf{A}$ into U portions with n/U rows and n columns. The u-th portion of $\mathbf{A}$ is called $\mathbf{A}_u$, where $u \in \{1, 2, \ldots, U\}$. By multiplying the u-th slice of $\mathbf{A}$ by $\mathbf{S}$, the corresponding u-th slice of $\mathbf{D}$, i.e., $\mathbf{D}_u$, of dimensions $n/U \times \overline{n}$ is obtained. By stacking all the slices $\mathbf{D}_u$, the matrix $\mathbf{D}$ (of size $n \times \overline{n}$) is formed.

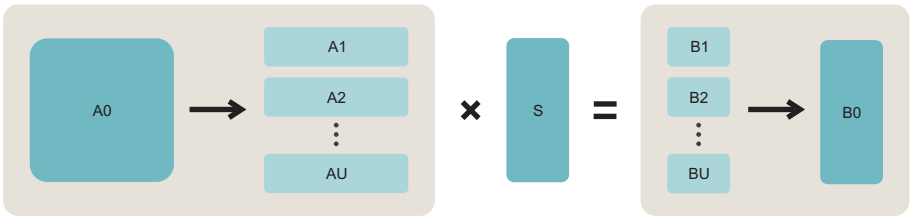


Figure 1. Illustration of the multiplication of $\mathbf{A}$ by $\mathbf{S}$ using U processing cores.

3.2. $\mathbf{S}'$ by $\mathbf{A}$ matrix multiplication

The multiplication of $\mathbf{S}'$ by $\mathbf{A}$ slightly differs from the multiplication of $\mathbf{A}$ by $\mathbf{S}$. In the former operation, the matrix $\mathbf{A}$ is placed on the right-hand side of the multiplication. However, the matrix $\mathbf{A}$ presented in (1) must be the same as that presented in (2) and (4), which means that it must be generated in the same way, row by row. On the other hand, as $\mathbf{A}$ is on the right-hand side, a column should be fully generated to perform the multiplication, which is not achieved using the tiled multiplication technique. Consequently, the same procedure reported in Subsection 3.1 cannot be used, requiring another matrix multiplication technique, see [Costa et al. 2022].

Therefore, we use another procedure for calculating the multiplication of $\mathbf{S}'$ by $\mathbf{A}$ using U processing cores. This procedure is illustrated in Figure 2. Note that the matrix $\mathbf{S}'$ is partitioned into U distinct column portions, where each portion, $\mathbf{S}'_u$, is assigned to the u-th processing core, with $u \in \{1, 2, \ldots, U\}$. Similarly, the matrix $\mathbf{A}$ is divided into U distinct row portions, with each portion, $\mathbf{A}_u$, assigned to the corresponding u-th processing core. Observe that the matrix $\mathbf{A}$ is generated row by row using SHAKE-128 and therefore the u-th processing core only needs to generate the rows associated with its corresponding portion, $\mathbf{A}_u$. Likewise, the u-th processing core only needs the u-th portion of matrix $\mathbf{S}'$ (i.e., $\mathbf{S}'_u$). The u-th processing core then computes the matrix multiplication of $\mathbf{S}'_u$ by $\mathbf{A}_u$, resulting in the matrix $\mathbf{D}'_u$. Finally, the sum of all matrices $\mathbf{D}'_u$ produces the matrix $\mathbf{D}'$.

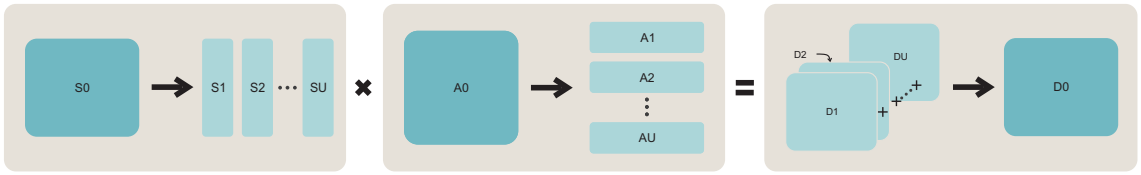


Figure 2. Illustration of the multiplication of $\mathbf{S}'$ by $\mathbf{A}$ using U processing cores.

3.3. Other matrices multiplication

Encapsulation and decapsulation of FrodoKEM involve additional matrix multiplications, such as the ones presented in (3). Since these operations do not use matrix $\mathbf{A}$, their matrix multiplications can be optimized using a multicore approach based on the strategy

presented in Subsection 3.1. The key difference is that $\mathbf{A}$ does not need to be generated and the involved matrices are already pre-generated. Therefore, there is no evident need to use the tiled multiplication technique. Instead, multicore multiplication and loop unrolling can achieve better performance.

4. Practical implementation

To analyze the execution time performance of the proposed multicore implementation of FrodoKEM, a practical implementation was performed using a hardware-constrained microcontroller. In this context, Subsection 4.1 addresses the particularities of the chosen microcontroller, an ESP32-S3. In Subsection 4.2, the feasibility of embedding FrodoKEM in the ESP32-S3 and the techniques required to make it viable are investigated. Finally, Subsection 4.3 discusses details of a dual-core implementation of FrodoKEM.

4.1. Main components

The hardware used in the implementation was an Espressif ESP32-S3 system-on-a-chip (SoC) [Espressif 2023], which is a dual-core Xtensa 32-bits LX7 microcontroller with 512 kB of SRAM memory. The chip also has integrated Wi-Fi 2.4 GHz (802.11 b/g/n) and Bluetooth Low Energy. Software development for the ESP32-S3 uses C programming language and the ESP-IDF framework. Additionally, it includes a modified FreeRTOS[2] implementation for dual-core support, enabling tasks to be pinned to specific core. Furthermore, FreeRTOS resources, such as semaphores, were also used for dual-core implementation. In this work, ESP-IDF version 5.1.4 was used.

4.2. Feasibility analysis

As previously discussed, the size of matrix $\mathbf{A}$ in FrodoKEM is significantly large. This makes it impractical to embed FrodoKEM as it is, even the lower security level variant FrodoKEM-640, in the ESP32-S3 microcontroller due to its limited memory capacity. It occurs because the ESP32-S3 provides 512kB of SRAM, while FrodoKEM-640 requires 819kB for storing matrix $\mathbf{A}$ alone. Therefore, the tiled multiplication technique must be employed. This implementation applied the tiled multiplication technique alongside a four-row loop unrolling. Consequently, each block (tile) has four rows of matrix $\mathbf{A}$. Table 2 lists the memory required to store each block (tile) of matrix $\mathbf{A}$ using the tiled multiplication technique.

Table 2. Size of matrix $\mathbf{A}$ and memory resource usage by matrix $\mathbf{A}$ in reference implementation and implementations with tiled multiplication technique.

	Size of matrix A		**Memory resource usage**	
	Reference	Tiled multiplication	Reference	Tiled multiplication
FrodoKEM-640	819.0 kB	5.12 kB	160%	1%
FrodoKEM-976	1.8 MB	7.80 kB	372%	1.5%
FrodoKEM-1344	3.6 MB	10.75 kB	705%	2.1%

[2]FreeRTOS is a real-time operating system that enables multitasking on limited hardware through preemptive scheduling.

Although the tiled multiplication technique does not enhance execution time performance, it substantially reduces the memory footprint, see Table 2. By applying this technique, the memory resource usage of matrix $\mathbf{A}$ in ESP32-S3 was reduced to around 2.1% in the worst case, enabling operations where matrix $\mathbf{A}$ is required and leaving more memory resources for other tasks.

4.3. Dual-core implementation details

Aiming to prove the performance improvement of the proposed multicore implementation of FrodoKEM, a dual-core implementation is embedded in the ESP32-S3. The proposed implementation aims to take advantage of the dual-core available in this microcontroller (i.e., $U = 2$) based on a modified FreeRTOS by ESP-IDF. When matrix multiplications are required, two tasks are created and pinned to each core. Each function executes half of the operations needed for the matrices using the tiled multiplication technique alongside loop unrolling. A structure is sent to each task containing pointers of data to be processed.

All operations were performed using memory pointers given the RAM limitations of the hardware and the pursuit of better performance. Thus joining all blocks (tiles) of the output matrix was unnecessary since the access to the output variable is shared between the tasks. However, another concern with this strategy was simultaneous access to the same memory location. In cases where the matrix $\mathbf{A}$ is on the left-hand side of the matrix product, it is not necessary to manage access because each task will write only the block (tile) of memory corresponding to the u-th block (tile), i.e., $\mathbf{D}_u$, as indicated in Figure 1. Since each value of u is unique, each processing core will always access different memory locations. On the other hand, if $\mathbf{A}$ is on the right-hand side, then multiple processing cores may access the same memory location of the matrix $\mathbf{D}'$ (or $\mathbf{D}''$) to sum its corresponding portion, i.e., $\mathbf{D}'_u$ (or $\mathbf{D}''_u$). Consequently, it is necessary to implement a mutal exclusion (mutex) lock strategy, which consists of controlling access to the memory blocks corresponding to the rows of the matrix $\mathbf{D}'$ (or $\mathbf{D}''$).

To address the issue in multiplying $\mathbf{S}'$ by $\mathbf{A}$, the authors propose that each processing core starts computing $\mathbf{D}'_u$ from a different row. To implement this, memory access coordination can be effectively managed within the matrix $\mathbf{S}'$, with data acquisition for matrices $\mathbf{S}'_u$ starting at different rows based on the processing core index u. Specifically, the u-th processing core begins acquiring data from $\mathbf{S}'_u$ at row index $(u-1)n/U+1$. In other words, the first processing core ($u = 1$) starts at the 1-st row of the matrix $\mathbf{S}'_1$, the second processing core ($u = 2$) starts at the $(n/U+1)$-th row of the matrix $\mathbf{S}'_2$, the third processing core ($u = 3$) starts at the $(2n/U+1)$-th row of the matrix $\mathbf{S}'_3$, and so on. Note that all processing cores must access all n rows of their respective matrices $\mathbf{S}'_u$, for all $u \in \{1, 2, \ldots, U\}$. Consequently, a cyclic behavior in row scanning is introduced: Upon reaching the last row n of $\mathbf{S}'_u$, each processing core returns to the first row and continues until the row before the one where it started. Figure 3 illustrates this process.

5. Numerical Results

This section evaluates the performance improvement obtained from a dual-core implementation of FrodoKEM on the ESP32-S3 microcontroller. We compare execution times for key generation, encapsulation, and decapsulation across the FrodoKEM-640, -976, and -1344 parameter sets. Each function was executed 101 times, and the median times are reported.

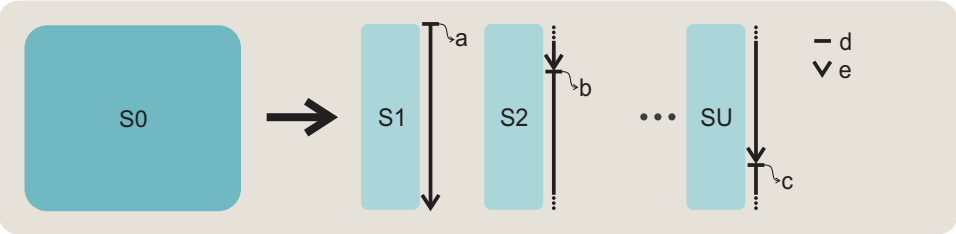


Figure 3. Illustration of the memory access coordination to the matrix S'.

Table 3. Execution time, in seconds, of FrodoKEM algorithms implemented in single- and dual-core of the ESP32-S3 microprocessor.

Description			Frodo-640	Frodo-976	Frodo-1344
Execution time (s)	Single-core	Key Pair Generation	2.20	5.10	9.35
		Encapsulation	2.45	5.63	10.46
		Decapsulation	2.45	5.65	10.47
		Total	**7.10**	**16.38**	**30.29**
	Dual-core	Key Pair Generation	1.12	2.59	4.75
		Encapsulation	1.40	2.91	5.75
		Decapsulation	1.41	2.91	5.77
		Total	**3.94**	**8.42**	**16.28**
Performance Improvement (%)		Key Pair Generation	48.8	49.1	49.2
		Encapsulation	42.8	48.4	45.0
		Decapsulation	42.3	48.4	44.9
		Total	**44.5**	**48.6**	**46.3**

As shown in Table 3, execution time increases with the matrix size, which is expected given that matrix $\mathbf{A}$ operations represent the main computational bottleneck. The dual-core implementation consistently outperformed the single-core version, achieving more than 44% reduction in execution time across all parameter sets and functions.

Ideally, data parallelism across two cores should yield a 50% speedup. In practice, the ESP32-S3 microcontroller incurs overhead from memory access and system-level tasks, preventing full parallel efficiency. Still, the results demonstrate that distributing matrix operations across cores is highly beneficial. For instance, FrodoKEM-1344 dropped from 30.29s to 16.28s, while FrodoKEM-640 was reduced from 7.10s to 3.94s.

These execution times show that FrodoKEM, even in its most computationally demanding version, can be deployed on resource-constrained devices when operating under relaxed latency constraints. This is particularly relevant for IoT applications such as smart metering and smart water systems, where data is typically exchanged in intervals of minutes. For example, smart meters report consumption every 15 minutes, and water management systems may transmit parameters every few minutes.

Moreover, reducing cryptographic processing time is critical in battery-powered IoT devices. Faster key establishment shortens the active processing period, allowing devices to return to low-power sleep modes more quickly, thereby improving energy effi-

ciency and extending operational lifespan.

6. Conclusions

This work demonstrated the viability of deploying FrodoKEM on hardware-constrained microcontrollers by combining tiled multiplication and data parallelism. Tiled multiplication enabled execution within memory limits, while data parallelism significantly reduced execution time. A dual-core implementation on the ESP32-S3 achieved a 44.5% speedup over the single-core version, with execution times ranging from 3.94 to 16.28 seconds. These results confirm the feasibility of using FrodoKEM in near real-time IoT environments requiring post-quantum security.

References

[Alkim et al. 2020] Alkim, E. et al. (2020). FrodoKEM: Learning with errors key encapsulation. Technical report, NIST, Gaithersburg, MD.

[Bos et al. 2023] Bos, J. W. et al. (2023). Enabling frodokem on embedded devices. *Cryptology ePrint Archive, Paper 2023/158*.

[Bos et al. 2021] Bos, J. W., Ofner, M., Renes, J., Schneider, T., and van Vredendaal, C. (2021). The matrix reloaded: Multiplication strategies in frodokem. In Conti, M., Stevens, M., and Krenn, S., editors, *Cryptology and Network Security*, pages 72–91, Cham. Springer International Publishing.

[BSI 2020] BSI, C. M. (2020). Recommendations and key lengths. Technical report, BSI.

[Costa et al. 2022] Costa, V. L. R. d., López, J., and Ribeiro, M. V. (2022). A system-on-a-chip implementation of a post-quantum cryptography scheme for smart meter data communications. *Sensors*, 22(19).

[Espressif 2023] Espressif (2023). ESP32-S3 series datasheet v1.8. Technical report, Espressif Systems.

[Karl et al. 2022] Karl, P., Fritzmann, T., and Sigl, G. (2022). Hardware accelerated frodokem on risc-v. In *2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pages 154–159.

[Kwon et al. 2021] Kwon, H., Jang, K., Kim, H., Kim, H., Sim, M., Eum, S., Lee, W.-K., and Seo, H. (2021). Armed frodo. In Kim, H., editor, *Information Security Applications*, pages 206–217, Cham. Springer International Publishing.

[Lee et al. 2022] Lee, W.-K., Seo, H. J., Seo, S. C., and Hwang, S. O. (2022). Efficient implementation of aes-ctr and aes-ecb on gpus with applications for high-speed frodokem and exhaustive key search. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 69(6):2962–2966.

[Regev 2010] Regev, O. (2010). The learning with errors problem (invited survey). In *Proc. IEEE 25th Annual Conference on Computational Complexity*, pages 191–204.

[Shor 1994] Shor, P. W. (1994). Algorithms for quantum computation: Discrete logarithms and factoring. In *Proc. 35th Annual Symposium on Foundations of Computer Science*, pages 124–134.

Improved Decryption Bounds and Key Generation for Matrix NTRU over Integral Domain

Thiago do Rêgo Sousa [1], Tertuliano Souza Neto[1]

[1] Centro de Pesquisa e Desenvolvimento para a Segurança das Comunicações (Cepesc)

thiagodoregosousa@gmail.com, tsouzaneto@gmail.com

Abstract. *Shor's algorithm [Shor 1994] is the main threat to classical public-key cryptography. Since its introduction in 1996, NTRU and its variants aim to develop cryptographic algorithms that are secure even against quantum computers. In this work, we study the matrix NTRU over integral domains proposed in 2023. We found that there is an error on the condition to avoid decryption failures and the key generation process is not practical due to severe limitations on matrix inversion. We propose a corrected statement for the decryption failure theorem and an expansion of the set of solutions when dealing with the problem of inverting matrix in* $M_n(\mathbb{Z}[\sqrt{-3}])$ *that makes the key generation significantly faster.*

1. Introduction

Shor's algorithm [Shor 1994] is the main threat to classical public-key cryptography. Since it can efficiently factor large integers and compute discrete logarithms, Shor's algorithm leads directly to attacks against well known and widely used cryptosystems such as RSA [Rivest et al. 1978] and Eliptic Curve Cryptography ([Koblitz 1987], [Miller 1985]).

The combination of Shor's algorithm and quantum computers, as well as the application of Grover's algorithm [Grover 1996], led the cryptographic community to develop what is called today's post-quantum cryptography (PQC). Since then, designing robust cryptographic schemes underpinned on mathematical problems that are intractable by both quantum and conventional computers is a fast growing research topic and the development of PQC has taken off in the last decade.

The National Institute of Standards and Technology (NIST) launched a PQC standardization project in 2016 [NIST]. The competition aimed to select and standardize cryptographic algorithms that are secure even against quantum computers. After four rounds and six years, NIST selected one algorithm for public-key encryption/KEM and three algorithms for digital signature. Two lattice-based systems have progressed significantly in the NIST Post-Quantum Cryptography standardization process [NIST]: the NTRU [Chen et al. 2020] and the NTRU Prime systems [Daniel J. Bernstein et al. 2024], both based on the original NTRU system from [Hoffstein et al. 1998].

Lattice-Based Cryptography relies on the difficulty of solving lattice-related problems, which are believed to be hard to solve even for a quantum computer. Among those problems, we have the NTRU-related ones like SVP (Shortest Vector Problem) and CVP (Closest Vector Problem), both believed to be quantum-resistant.

Introduced at CRYPTO'96, the NTRU cryptosystem has since undergone extensive analysis and numerous extensions. These include variations in polynomial

coefficients and the adoption of matrix-based structures. Comprehensive reviews can be found in works such as [Singh and Padhye 2016, Salleh and Kamarulhaili 2020, Mittal and Ramkumar 2022]

Regarding NTRU and the NIST PQC standardization process, a variant called NTRUEncrypt and its relatives were submitted and NTRUEncrypt was selected as an alternate candidate, which means that it wasn't standardized at first, but it is considered valuable and can still be used or standardized later. The growing interest on NTRU algorithm is because it has interesting properties, such as speed and small key sizes compared to some other PQC algorithms, resistance to both classical and quantum attacks, and easy adaptation into newer variants.

Since the introduction of NTRU, many variants have been proposed in the literature [Salleh and Kamarulhaili 2020]. The first NTRU generalization using matrices was introduced in [Coglianese and Goi 2005], where key generation, encryption and decryption operate over matrices whose entries are polynomials. In 2008, [Nayak et al. 2008] proposed a matrix-based system as a variant of NTRU using only matrices with integer entries, replacing the arithmetic of truncated polynomials with a simple modular arithmetic on matrices. This new system, called matrix NTRU, was subject to some further analysis and even suggested for real data applications. However, [Sousa and Neto 2024] showed that Matrix NTRU is completely insecure and should be avoided in practical applications.

More recently, [Wijayanti et al. 2023] developed a variant of NTRU which is based on matrix arithmetic defined over $\mathbb{Z}[\sqrt{-3}]$ called Matrix NTRU over Integral Domains. We can say that this system is a Matrix NTRU variant, where the authors replaced the underlined ring of the entries of the matrices. Of course they had to make some changes in the key generation process and the approach of decryption failures.

When diving into mathematical details and in implementing [Wijayanti et al. 2023]'s cryptosystem, we found out that there is an error on the condition to avoid decryption failures and the key generation process is not practical due to severe limitations on matrix inversion.

Our main contribution to matrix NTRU over integral domain is twofold. First we made a more intensive simulation study and found a counterexample to [Wijayanti et al. 2023, Theorem 2] and these results, together with [Sousa and Neto 2024, Proposition 1], led us to propose a corrected statement for the decryption failure theorem. Second, we found that the conditions from the proof of [Wijayanti et al. 2023, Theorem 1] used in their paper for the key generation process are overly restrictive making it very inefficient (around 14 minutes to generate keys for matrix of dimension 7×7). To solve this problem we propose the expansion of the set of solutions when dealing with the problem of inverting matrix in $M_n(\mathbb{Z}[\sqrt{-3}])$ lowering the time for key generation of 7×7 matrices to less than one second.

The rest of the paper is organized as follows. In Section 2, we give some necessary background on Algebraic Number Theory with respect to modular algebra over matrices as used in [Wijayanti et al. 2023]. In Section 3 we show how the algorithm works in details and discuss their decryption failure theorem and the limitations of the key generation process. In Section 4 we give a detailed example that abides to the conditions of [Wijayanti et al. 2023, Theorem 2] to avoid decryption failure, but the ciphertext does not

correctly decrypts to the original message. Then we correct the decryption failure condition and make an intensive simulation study on critical parameters to support the result. In Section 5 we dive deeper into the conditions for key generation, propose an extension of the solution space for finding matrix inverses and conclude with an experimental study showing that this extended process not only works but is significantly faster than [Wijayanti et al. 2023]'s original algorithm. Finally, we conclude in Section 6 and give further research directions.

2. Background

Let's consider the ring

$$\mathbb{Z}[\sqrt{-3}] = \{a + b\sqrt{-3}, a, b \in \mathbb{Z}\}.$$

From Algebraic Number Theory, it is well known that $\mathbb{Z}\sqrt{-3}]$ is an integral domain. For any $\alpha = a + b\sqrt{-3} \in \mathbb{Z}[\sqrt{-3}]$ and p a prime number, we define

$$\alpha \mod p = (a \mod p) + (b \mod p)\sqrt{-3}.$$

We define $M_n(\mathbb{Z}[\sqrt{-3}])$ as the set of $n \times n$ matrices whose coefficients belong to $\mathbb{Z}[\sqrt{-3}]$.

As is common in cryptosystems like NTRU and its variants, we are going to work with ternary elements. In the case of Matrix NTRU over Integral Domain being ternary simply means that the real and imaginary parts of its entries are in $\in \{-1, 0, 1\}$.

Let p be a prime number and let

$$A = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix}$$

be a matrix in $M_n(\mathbb{Z}[\sqrt{-3}])$. Reduction modulo p in $M_n(\mathbb{Z}[\sqrt{-3}])$ is quite straightforward. We say that A is reduced modulo p, denoted as $A \mod p$, if every entry in A is reduced modulo p. Therefore

$$A \mod p = \begin{pmatrix} a_{11} \mod p & \dots & a_{1n} \mod p \\ \vdots & \ddots & \vdots \\ a_{n1} \mod p & \dots & a_{nn} \mod p \end{pmatrix}. \tag{1}$$

We can also define sum and multiplication of matrices modulo p. Let A and B be matrices in $M_n(\mathbb{Z}[\sqrt{-3}])$. Operation $A + B \mod p$ is done component-wise, so we can define

$$(A + B) \mod p = (A \mod p + B \mod p) \mod p.$$

In the same way, multiplication is defined as

$$(A * B) \mod p = (A \mod p * B \mod p) \mod p.$$

A careful reader will notice that the modulo p operation is not always canonical. That is common in NTRU-like cryptosystems, because of what is called center lift operation, which is necessary for the decryption process work properly. In that case, it can be useful to consider a non-canonical representation of the elements on the rings on what the coefficients of the matrices belong to. Therefore, when we are dealing with a prime modulus, we have

$$\mathbb{F}_p = \left\{ -\frac{p-1}{2}, \ldots, \frac{p-1}{2} \right\}$$

and for a composite number, we have

$$\mathbb{Z}_q = \left\{ -\frac{q}{2} + 1, \ldots, \frac{q}{2} \right\}.$$

Since modular arithmetic over the rings $M_n(\mathbb{Z}[\sqrt{-3}])$ is crucial for the Matrix NTRU cryptosystem over Integral Domain work properly, we need to address the question of how to invert matrices modulo p in that ring. First of all, a matrix A in this ring is invertible modulo p if and only if p and the determinant of A are relatively prime [Wijayanti et al. 2023]. If that is the case, then there exists a (unique) matrix B such that $AB = BA = I_n$ in $M_n(\mathbb{Z}[\sqrt{-3}])$, where I_n is the $n \times n$ identity matrix. The matrix B is called the inverse of A modulo p and denoted $A^{-1} \mod p$.

3. Matrix NTRU Cryptosystem over Integral Domain

3.1. How the algorithm works

As we already know, Matrix NTRU over Integral Domain cryptosystem is defined over matrices whose coefficients are in the ring $\mathbb{Z}[\sqrt{-3}]$. We have also noticed that we have to deal with the question on how to invert matrices in a correct and efficient way.

Let $A \in M_n(\mathbb{Z}[\sqrt{-3}])$ and suppose $\det(A) \neq 0$. If $\gcd(\det(A), p) = 1$, then A is invertible modulo p and we can write

$$A^{-1} \mod p = (\det(A))^{-1} \mathrm{Adj}(A),$$

where $\mathrm{Adj}(A)$ represents the adjoint matrix of A. The previous result is a classical one regarding matrix theory and the interested reader can see [Hungerford 2012] for a proof.

The method that can be used to invert the matrices is really important if one wants to develop a secure and efficient key generation process. The authors in [Wijayanti et al. 2023] modify the algorithm in the original development of Matrix NTRU cryptosystem [Nayak et al. 2008] to replace the condition for matrix inversion. Although their choice brought interesting properties, we show that it has serious limitations with a huge impact on the speed of key generation, as we will show later in this work.

In what follows, let $\mathcal{L}(d)$ be the set of all matrices in $M_n(\mathbb{Z}[\sqrt{-3}])$ with d coefficients equal to 1, d coefficients equal to -1 and $n^2 - 2d$ entries equal to zero. In [Wijayanti et al. 2023], they take the values for d near $\frac{2n^2}{3}$ and we follow them to keep our results comparable to theirs. As an example, we use $d = 5$ for $n = 3$ just like them for some specific experiments here, although the precise d value should be 6 according to the formula.

Matrix NTRU over Integral Domains works as follows.

1. Key Generation: Let p and q be prime numbers. The private and public keys are $n \times n$ matrices randomly chosen from the set $\mathcal{L}(d)$. First, we choose a pair of ternary matrices F, G such that $F \mod p$ and $F \mod q$ are invertible. Then we compute the matrices $F_p = F^{-1} \mod p$ and $F_q = F^{-1} \mod q$. The parameters F, G, F_p, F_q should be kept secret. Now, we can compute the public key by performing the calculation

$$H = pF_qG \mod q.$$

2. Encryption: To encrypt a message, we encode it as a matrix M whose coefficients are in the form $a + b\sqrt{-3}$ with $a, b \in \mathbb{F}_p$. Next, we generate a random ternary matrix $R \in M_n(\mathbb{Z})$. Now, we compute the ciphertext as

$$E = HR + M \mod q.$$

3. Decryption: To decrypt, we compute

$$A = FE \mod q$$

applying the center-lift operation, which means that all A coefficients have the form $a + b\sqrt{-3}$ with $a, b \in \mathbb{F}_q$. Now we compute $B = F_pA \mod p$ and the calculation of

$$C = F_pB \mod p$$

gives the desired plaintext.

The reader who wants to be sure that the decryption algorithm works can take a look at [Wijayanti et al. 2023].

3.2. Decryption failure in detail and Key generation process

Depending on the parameters selected, there can be decryption errors when operating on Matrix NTRU over Integral Domains just like other NTRU variants cryptosystems and even NTRU itself. That can happen because matrices C and M, as defined on previous subsection, are not always equal and we say there is a decryption failure in that case. This is an important issue concerning any probabilistic cryptosystem since attackers can take advantage of it, recovering secret data by exploring those decryption failures (see [Jaulmes and Joux 2000] or [Gama and Nguyen 2007]). This justifies the importance of having sufficient conditions under which decryption failures are avoided.

We notice that the authors [Wijayanti et al. 2023] did address that question, although there are corrections to perform regarding decription failures on Matrix NTRU over Integral Domains.

Theorem 1 (Theorem 2 in [Wijayanti et al. 2023], Incorrect)**.** *If the parameters n, p, q, d satisfy*

$$q > 3\sqrt{7}np,$$

then the system decrypts correctly.

As we demonstrate in the next section, Theorem 1 is incorrect, as shown by a counterexample. Fortunately, we are able to correct the decryption failure condition and

validate it extensively, particularly near the boundary cases, to support the new claim alongside the mathematical proofs.

Regarding the key generation process, although the method the authors [Wijayanti et al. 2023] applied for matrix inversion is correct, we discovered that a slight modification can significantly accelerate the generation of the coefficients in Theorem 1 from [Wijayanti et al. 2023].

In a short summary, we could say that the authors in [Wijayanti et al. 2023] made a severe restriction when dealing with the generation of the coefficients on the Extended Euclidean Algorithm in the ring $\mathbb{Z}[\sqrt{-3}]$. We notice that the same is true regarding the determinant of the private key F. We show how to speed up the key generation by letting the coefficients α and β in Theorem 1 of [Wijayanti et al. 2023] take values in $\mathbb{Z}[\sqrt{-3}]$ as well as the determinant of the matrix F.

4. A Counterexample and Correction to the Decryption Failure Theorem

In such public key system it is advisable to provide necessary conditions under which decryption failure does not happen. An attempt to provide such condition was done in [Wijayanti et al. 2023] and is described in Theorem 1. For $n = 3, p = 3$ this condition boils down to $q > 71.44$. The authors present examples where they tested the values for $n = 3, p = 3$ and q varying from 13 to 673, but they tested the encryption and decryption process only 100 times for each case which was not enough to spot inconsistencies in their result. They spotted no errors for $q = 73$ which abides to the condition $q > 71.44$ but as we show in Example 4.1, $q = 73$ gives indeed decryption failure when tested more intensively.

Example 4.1 (Counterexample to Theorem 1)**.** *This is an example of a decryption failure for $N = 3, p = 3, q = 73, d = 5$, where the chosen parameters satisfy the theorem's hypotheses and should, in principle, permit correct decryption of the ciphertext. However, as we detail below, this does not occur. The private keys chosen are F and G:*

$$F = \begin{bmatrix} 0-\sqrt{-3} & 0 & -1 \\ -1 & 1-\sqrt{-3} & -1 \\ \sqrt{-3} & 1+\sqrt{-3} & 1 \end{bmatrix} \quad G = \begin{bmatrix} -\sqrt{-3} & -1+\sqrt{-3} & 0 \\ 0 & -1 & 1 \\ -\sqrt{-3} & 1-\sqrt{-3} & 1+\sqrt{-3} \end{bmatrix}$$

The corresponding inverses F_p and F_q are:

$$F_p = \begin{bmatrix} 2 & 2+2\sqrt{-3} & 2+\sqrt{-3} \\ 1+2\sqrt{-3} & 0 & 1+2\sqrt{-3} \\ 2+\sqrt{-3} & \sqrt{-3} & 2\sqrt{-3} \end{bmatrix}$$

and

$$F_q = \begin{bmatrix} 37 & 18+18\sqrt{-3} & 55+18\sqrt{-3} \\ 55+18\sqrt{-3} & 0 & 55+18\sqrt{-3} \\ 72+36\sqrt{-3} & 54+55\sqrt{-3} & 54+18\sqrt{-3} \end{bmatrix}$$

The public key is $H = pF_qG$:

$$H = \begin{bmatrix} 16+16\sqrt{-3} & 19+16\sqrt{-3} & 57+54\sqrt{-3} \\ 32+35\sqrt{-3} & 0 & 3 \\ 48+60\sqrt{-3} & 60+54\sqrt{-3} & 16+16\sqrt{-3} \end{bmatrix}$$

Consider we want to encrypt the message M using a random matrix R:

$$M = \begin{bmatrix} -\sqrt{-3} & \sqrt{-3} & 1-\sqrt{-3} \\ 1 & -1+\sqrt{-3} & -1 \\ -\sqrt{-3} & \sqrt{-3} & 0 \end{bmatrix} \quad R = \begin{bmatrix} \sqrt{-3} & -1-\sqrt{-3} & \sqrt{-3} \\ \sqrt{-3} & -1-\sqrt{-3} & -1 \\ 1 & \sqrt{-3} & 0 \end{bmatrix}$$

The corresponding ciphertext is computed as $E = pHR + M \pmod q$:

$$E = \begin{bmatrix} 25+12\sqrt{-3} & 57+64\sqrt{-3} & 10+69\sqrt{-3} \\ 45+32\sqrt{-3} & 72+10\sqrt{-3} & 40+32\sqrt{-3} \\ 39+50\sqrt{-3} & 40+14\sqrt{-3} & 52+67\sqrt{-3} \end{bmatrix}$$

To decrypt, we first compute $A = FE \pmod q$:

$$A = \begin{bmatrix} 70+71\sqrt{-3} & 6+2\sqrt{-3} & 9+69\sqrt{-3} \\ 4+71\sqrt{-3} & 5+6\sqrt{-3} & 1+2\sqrt{-3} \\ 25+6\sqrt{-3} & 36+7\sqrt{-3} & 8+3\sqrt{-3} \end{bmatrix}$$

and

$$A_{\textit{center\_lift}} = \begin{bmatrix} -3-2\sqrt{-3} & 6+2\sqrt{-3} & 9-4\sqrt{-3} \\ 4-2\sqrt{-3} & 5+6\sqrt{-3} & 1+2\sqrt{-3} \\ 25+6\sqrt{-3} & 36+7\sqrt{-3} & 8+3\sqrt{-3} \end{bmatrix}$$

Then we compute $B = A \pmod p$:

$$B = \begin{bmatrix} \sqrt{-3} & 2\sqrt{-3} & 2\sqrt{-3} \\ 1+\sqrt{-3} & 2 & 1+2\sqrt{-3} \\ 1 & \sqrt{-3} & 2 \end{bmatrix} \quad B_{\textit{center\_lift}} = \begin{bmatrix} \sqrt{-3} & -\sqrt{-3} & -\sqrt{-3} \\ 1+\sqrt{-3} & -1 & 1-\sqrt{-3} \\ 1 & \sqrt{-3} & -1 \end{bmatrix}$$

Finally, we compute $C = F_pB \pmod p$:

$$C = \begin{bmatrix} 2\sqrt{-3} & 1 & 1+2\sqrt{-3} \\ 1 & 0 & 2 \\ 2\sqrt{-3} & 0 & 0 \end{bmatrix} \quad C_{\textit{center\_lift}} = \begin{bmatrix} -\sqrt{-3} & 1 & 1-\sqrt{-3} \\ 1 & 0 & -1 \\ -\sqrt{-3} & 0 & 0 \end{bmatrix}$$

Note that $C \neq M$.

We found that [Wijayanti et al. 2023] did not took into account the fact that decryption failures can occur if any real or complex component of the matrices involved during decryption is not smaller than $q/2$.

A corrected statement of a necessary condition for decryption failure will be given in the next Section which also includes further experimental investigation of boundary cases to strengthen and test this new condition. The corrected condition requires $q > 96$, i.e., the minimum value of q to create a system without decryption failure is $q = 97$.

4.1. Correcting the decryption failure condition

Theorem 2. *Suppose that the system parameters n, p, q satisfy*

$$q > 8n\left(p + \left\lceil \frac{p-1}{2} \right\rceil\right).$$

Then the system decrypts correctly.

Proof. We first determine the shape of the calculation of A during decryption. Considering that

$$A \equiv pGR + FM(\mod q)$$

we need to restrict its largest possible entries of $pGR + FM$.

More specifically, none of the components (real or imaginary) of any entry of the above matrix should exceed $q/2$ in order for the decryption to work (see [Silverman et al. 2008, Proposition 6.48]). This is a direct consequence of the fact that in the matrix NTRU over integral domain, the module q operation for matrices was defined component-wise in (1).

Since the product of the matrices involves products of elements in $Z(\sqrt{-3})$ we start by considering the sets

$$\mathbb{T}_1 = \{-1, 0, 1\}$$

and

$$\mathbb{T}_k = \{-k, \ldots, 0, \ldots, k\}, \quad k \in \mathbb{Z}.$$

Given arbitrary $\alpha = a + b\sqrt{-3}, \beta = c + d\sqrt{-3} \in \mathbb{Z}[\sqrt{-3}]$ we have

$$\alpha\beta = (a + b\sqrt{-3})(c + d\sqrt{-3}) = ac - 3bd + (ad + bc)\sqrt{-3},$$

If $a, b, c, d \in \mathbb{T}_1$ then

$$\max\{ac - 3bd\} = 4 \quad \text{and} \quad \max\{ad + bc\} = 2$$

so the maximum absolute value of the entries of $\alpha\beta$ in this scenario is 4. In a similar way, If $a, b \in \mathbb{T}_1$ and $c, d \in \mathbb{T}_k$ then

$$\max\{ac - 3bd\} = k - 3(-k) = 4k \quad \text{and} \quad \max\{ad + bc\} = k + k = 2k$$

so the maximum absolute value of the entries of $\alpha\beta$ in this second scenario is $4k$.

Now we will bound the largest value of the components of the matrices involved in

$$A \equiv pGR + FM \mod q.$$

Since both G and R are in $\mathcal{L}$, their entries are elements whose components are in $\mathbb{T}_1$ and therefore the largest entry of pGR is $4np$. Now for the product FM, the entries of F have components in $\mathbb{T}_1$ but the entries of M have components in $\mathbb{T}_k$ with $k = \lceil \frac{p-1}{2} \rceil$. Therefore, by the same argument we see that the largest entry is $n(4\lceil \frac{p-1}{2} \rceil)$.

Putting it all together, the largest entry in $pGR + FM$ is $4n(p + k)$ and since it should be smaller than $q/2$ ([Silverman et al. 2008, proof of Proposition 6.48]) the decryption failure condition boils down to

$$q/2 > 4n\left(p + \left\lceil \frac{p-1}{2} \right\rceil\right),$$

which completes the proof. □

For $p = 3, n = 3$ the condition is $q > 96$ which differs from the one found by [Wijayanti et al. 2023] ($q > 71.44$ and it is now clear why the previous incorrect condition would be prone to decryption failure.

4.2. Experimental Support for Theorem 2 Correctness in Boundary Scenarios

In order to assess empirically the correctness of our new decryption failure Theorem 2 it is crucial to test it at the boundary, which the decryption failure condition is tested more often. Fixing $p = 3, n = 3$ in Theorem 2 gives the condition $q > 96$. Since q must be prime, the values $q = 89$ and $q = 97$ are particularly interesting: for $q = 89$, decryption is expected to fail at some point, while for $q = 97$, no failure should occur. The example below illustrates the case $q = 89$, listing all matrices generated that satisfy the system's hypotheses and demonstrating that decryption fails. For $q = 97$ we do intensive encryption and decryption testing.

Example 4.2 (Testing boundary values of Theorem 2)**.** *Let $p = 3, n = 3, q = 89, d = 5$. The value $q = 89$ corresponding to a prime slightly smaller than the required restriction $q > 96$, which tell us there might be a decryption failure. The private keys chosen are*

$$F = \begin{bmatrix} -1-\sqrt{-3} & 1-\sqrt{-3} & -1-\sqrt{-3} \\ 1 & \sqrt{-3} & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

and

$$G = \begin{bmatrix} -1-\sqrt{-3} & 1-\sqrt{-3} & -1-\sqrt{-3} \\ 0 & 1 & 1 \\ \sqrt{-3} & 0 & 1 \end{bmatrix}.$$

The corresponding inverses F_p and F_q are:

$$F_p = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1+\sqrt{-3} & 0 \\ 2\sqrt{-3} & 1+2\sqrt{-3} & 2 \end{bmatrix} \quad F_q = \begin{bmatrix} 0 & 0 & 1 \\ 44 & 44+44\sqrt{-3} & 0 \\ 45\sqrt{-3} & 44+45\sqrt{-3} & 88 \end{bmatrix}$$

The public key is $H = pF_qG$:

$$H = \begin{bmatrix} 3\sqrt{-3} & 0 & 3 \\ 46+46\sqrt{-3} & 86 & 0 \\ 49+40\sqrt{-3} & 3+3\sqrt{-3} & 0 \end{bmatrix}$$

Consider we want to encrypt the message M using a random matrix R:

$$M = \begin{bmatrix} 1-\sqrt{-3} & 0 & 0 \\ -1-\sqrt{-3} & 0 & \sqrt{-3} \\ 1-\sqrt{-3} & -1+\sqrt{-3} & 1 \end{bmatrix} \quad R = \begin{bmatrix} 1-\sqrt{-3} & \sqrt{-3} & 1 \\ -1-\sqrt{-3} & 0 & -1 \\ 1-\sqrt{-3} & 1 & 0 \end{bmatrix}$$

The corresponding ciphertext is computed as $E = pHR + M \pmod q$:

$$E = \begin{bmatrix} 13+88\sqrt{-3} & 83 & 3\sqrt{-3} \\ 8+2\sqrt{-3} & 40+46\sqrt{-3} & 49+47\sqrt{-3} \\ 87+73\sqrt{-3} & 57+50\sqrt{-3} & 47+37\sqrt{-3} \end{bmatrix}$$

To decrypt, we first compute $A = FE \pmod q$:

$$A = \begin{bmatrix} 41 & 10+83\sqrt{-3} & 85 \\ 5+80\sqrt{-3} & 2+\sqrt{-3} & 84 \\ 13+88\sqrt{-3} & 83 & 3\sqrt{-3} \end{bmatrix}$$

and

$$A_{\textit{center\_lift}} = \begin{bmatrix} 41 & 10-6\sqrt{-3} & -4 \\ 5-9\sqrt{-3} & 2+\sqrt{-3} & -5 \\ 13-\sqrt{-3} & -6 & 3\sqrt{-3} \end{bmatrix}$$

Then we compute $B = A \pmod p$:

$$B = \begin{bmatrix} 2 & 1 & 2 \\ 2 & 2+\sqrt{-3} & 1 \\ 1+2\sqrt{-3} & 0 & 0 \end{bmatrix} \quad B_{\textit{center\_lift}} = \begin{bmatrix} -1 & 1 & -1 \\ -1 & -1+\sqrt{-3} & 1 \\ 1-\sqrt{-3} & 0 & 0 \end{bmatrix}$$

Finally, we compute $C = F_pB \pmod p$:

$$C = \begin{bmatrix} 1+2\sqrt{-3} & 0 & 0 \\ 1+2\sqrt{-3} & 0 & \sqrt{-3} \\ 1 & 2+\sqrt{-3} & 1 \end{bmatrix} \quad C_{\textit{center\_lift}} = \begin{bmatrix} 1-\sqrt{-3} & 0 & 0 \\ 1-\sqrt{-3} & 0 & \sqrt{-3} \\ 1 & -1+\sqrt{-3} & 1 \end{bmatrix}$$

Note that $C_{\textit{center\_lift}} \neq M$ (specifically at entries (2,1) and (2,2)).

The value for $q = 97$ is worth being intensively tested. We had crafted two scenarios. The first one has private keys, messages M and random R's crafted in order to maximaze the magnitude of the entry at position (1,1) of the matrix $pGR + FM$ to force a possible decryption failure. We tested 10^7 times and no decryption failure were found. We have also devised another experimental setting where both keys and messages were chosen randomly in $\mathcal{L}(d)$, and no decryption failure appeared after running the experiment 10^6 times.

We have also devised an experiment varying the values of n and d. The value of q was choosen to be the smallest prime satysfying the conditions of Theorem 2. The results are shown in Table 1.

Table 1. Decryption failure rate for the NTRU-like matrix scheme using the faster and more general key generation process. Here, q is the smallest prime satisfying Theorem 2, and d is approximately $\frac{2}{3}n^2$ as in [Wijayanti et al. 2023].

n	q	d	Number of Decryption Failure
3	97	6	0
4	131	11	0
5	163	17	0
6	193	24	0
7	227	33	0
10	331	67	0
15	487	150	0

5. Key Generation: Limitations, Corrections, and Efficient Approaches

5.1. Limitations of Existing Key Generation Methods and challenges

The key generation process proposed in [Wijayanti et al. 2023] for the matrix NTRU over integral domain require very strict conditions on the determinant of F in order for it to be invertible modulo p and q. They require the existence of integers α and β such that $\alpha\det(F) + \beta p = 1$, and similarly for q. This led to a very slow key generation process, which was reported in [Wijayanti et al. 2023, Table III], where key generation for $n = 7$ takes around 14 minutes on a personal computer.

The authors argue that such slow process is probably due to the non existence of the determinant inverse modulo p or q, but we show that if one allows both α and β to be in $\mathbb{Z}[\sqrt{-3}]$, then the inverse of F is not only fast to calculate but it exists in most cases. In fact, valid keys can be generated for $n = 7$ in 0.01 seconds, which significantly improves over the process proposed in [Wijayanti et al. 2023].

5.2. Efficient Key Generation Using the Extended Euclidean Algorithm in $\mathbb{Z}[\sqrt{-3}]$ and a simulation study of its efficiency

A critical step in the Matrix-NTRU cryptosystem over the ring $\mathbb{Z}[\sqrt{-3}]$ is the generation of a matrix $F \in M_n(\mathbb{Z}[\sqrt{-3}])$ that admits modular inverses modulo two distinct primes p and q. That is, we require $F^{-1} \bmod p$ and $F^{-1} \bmod q$ to exist in order to compute the private and public key pair. This condition depends directly on the invertibility of $\det(F)$ in the ring $\mathbb{Z}[\sqrt{-3}]$.

In [Wijayanti et al. 2023]'s approach, this invertibility was achieved by requiring the solution of the Diophantine equation

$$\alpha \det(F) + \beta p = 1$$

with $\alpha, \beta \in \mathbb{Z}$. This restriction dramatically impacts efficiency: since $\det(F)$ lies in $\mathbb{Z}[\sqrt{-3}]$, finding integer coefficients satisfying the above equation becomes increasingly unlikely as n grows, leading to impractical key generation times. For instance, for $n = 7$, it was reported that generating a valid matrix F could take over 10 minutes, severely limiting scalability (see [Wijayanti et al. 2023, Table III]).

To overcome this bottleneck, we propose lifting the restriction that α, β must be integers. Instead, we solve the extended Euclidean equation in the domain $\mathbb{Z}[\sqrt{-3}]$ itself.

That is, given a prime p, we apply the extended Euclidean algorithm in $\mathbb{Z}[\sqrt{-3}]$ to find $\alpha, \beta \in \mathbb{Z}[\sqrt{-3}]$ satisfying:

$$\alpha \det(F) + \beta p = 1.$$

When such α exists, we define the modular inverse of F modulo p as:

$$F^{-1} \bmod p = \alpha \text{adj}(F),$$

where $\text{adj}(F)$ denotes the adjugate matrix of F. In the case the equation resolves to -1, we take $-\alpha\text{adj}(F)$ as the modular inverse.

This generalization dramatically improves efficiency: most matrices F with nonzero determinants in $\mathbb{Z}[\sqrt{-3}]$ admit modular inverses modulo p and q under this extended notion. As a result, key generation becomes not only feasible but fast, even for larger values of n, as the experimental results in Table 2 demonstrate.

We performed a Simulation with Generalized Key Generation using parameters that abide to the Corrected Decryption Failure Bounds from Theorem 2. The results are shown in Table 2 where we see that new key generation not only outperforms the initial approach suggested in [Wijayanti et al. 2023] but makes it fast for moderate values of n. It is worth noting that the values of p, q did not significantly affect the running time of the system as reported in [Wijayanti et al. 2023] and therefore we think this comparisons makes sense.

Table 2. Comparison of key generation times (in seconds) between the reference method ([Wijayanti et al. 2023]) and the method presented in this work. Average time execution is presented for our work based on 1000 executions. The values for the computational time for key generation in [Wijayanti et al. 2023] are the correspondent ones presented in their Table III where it is not clear which p and q was used and therefore, we reported their minimum time for each dimension for a somewhat fair comparison.

n	q	d	KeyGen [Wijayanti et al. 2023] (s)	KeyGen THIS WORK (s)
3	97	6	0.17336	0.00327
4	131	11	0.93104	0.00495
5	163	17	7.33099	0.00719
6	193	24	83.7658	0.01045
7	227	33	766.048	0.01313
10	331	67	not available	0.03368
15	487	150	not available	0.11421

The most important lesson from Table 2 is that [Wijayanti et al. 2023]'s approach to generate keys for the matrix NTRU over the integral domain was overly restrictive making it hard to find keys even for moderate values of n, but this is immediately solved by using extended Euclidean algorithm in the integral domain $\mathbb{Z}[\sqrt{-3}]$.

6. Conclusion

We have studied the theory behind the new post-quantum system Matrix NTRU over integral domain and corrected an important result on its decryption failure condition. In

addition, we replaced the previous approach of key generation by one that is more practical and much faster for this purpose.

Although we have made significant strides in making the matrix NTRU over integral domain more practical it is important to note that we are not recommending its deployment at this stage. Future work should focus on a thorough analysis of its security. We believe that the system may still be vulnerable to attacks in a severe way, particularly through the breaking of the private key matrix in individual rows combined with an integer lattice attack using the isomorphism presented in [Monica Nevins 2010]. This is an interesting future research question. Source code is made available at `https://github.com/thiagopod17/integral_domain_matrix_ntru`.

References

Chen, C., Danba, O., Hoffstein, J., Hulsing, A., Rijneveld, J., Schanck, J. M., Schwabe, P., Whyte, W., and Zhang, Z. (2020). Ntru: algorithm specifications and supporting documentation (2019). *URL: https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.*, 1.

Coglianese, M. and Goi, B.-M. (2005). Matru: A new ntru-based cryptosystem. In *Progress in Cryptology-INDOCRYPT 2005: 6th International Conference on Cryptology in India, Bangalore, India, December 10-12, 2005. Proceedings 6*, pages 232–243. Springer.

Daniel J. Bernstein, Tanja Lange, Chitchanok Chuengsatiansup, and Peter Schwabe (Accessed: 2024). NTRU Prime. `https://ntruprime.cr.yp.to`. Website.

Gama, N. and Nguyen, P. Q. (2007). New chosen-ciphertext attacks on ntru. In *International Workshop on Public Key Cryptography*, pages 89–106. Springer.

Grover, L. K. (1996). A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219.

Hoffstein, J., Pipher, J., and Silverman, J. H. (1998). Ntru: A ring-based public key cryptosystem. In *International algorithmic number theory symposium*, pages 267–288. Springer.

Hungerford, T. W. (2012). *Algebra*, volume 73. Springer Science & Business Media.

Jaulmes, É. and Joux, A. (2000). A chosen-ciphertext attack against ntru. In *Annual international cryptology conference*, pages 20–35. Springer.

Koblitz, N. (1987). Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209.

Miller, V. S. (1985). Use of elliptic curves in cryptography. In *Conference on the theory and application of cryptographic techniques*, pages 417–426. Springer.

Mittal, S. and Ramkumar, K. (2022). A retrospective study on ntru cryptosystem. In *AIP Conference Proceedings*, volume 2451. AIP Publishing.

Monica Nevins, Camelia KarimianPour, A. M. (2010). Ntru over rings beyond z. *Designs, Codes and Cryptography*, 56:65–78.

Nayak, R., Sastry, C., and Pradhan, J. (2008). A matrix formulation for ntru cryptosystem. In *2008 16th IEEE International Conference on Networks*, pages 1–5. IEEE.

NIST. Post-Quantum Cryptography Standardization. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization.

Rivest, R. L., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126.

Salleh, N. and Kamarulhaili, H. (2020). Ntru public-key cryptosystem and its variants: An overview. *Int. l J. of Cryptology Research*, 10(1):1–21.

Shor, P. W. (1994). Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee.

Silverman, J. H., Pipher, J., and Hoffstein, J. (2008). *An introduction to mathematical cryptography*, volume 1. Springer.

Singh, S. and Padhye, S. (2016). Generalisations of ntru cryptosystem. *Security and Communication Networks*, 9(18):6315–6334.

Sousa, T. and Neto, T. S. (2024). Lattice base reduction attack on matrix ntru. In *Anais do XXIV Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, pages 431–444, Porto Alegre, RS, Brasil. SBC.

Wijayanti, I. E., Isnaini, U., Sari, A. K., Ali, S., and Aji, N. C. (2023). Matrix ntru cryptosystem over integral domain. In *2023 IEEE International Conference on Cryptography, Informatics, and Cybersecurity (ICoCICs)*, pages 35–40. IEEE.

Multicore Implementation of ML-KEM on Embedded Devices

Beatriz L. Azevedo[1], **Vinícius Lagrota**[2], **Moisés V. Ribeiro**[1]

[1] Electric Engineering Department
Federal University of Juiz de Fora (UFJF) – Juiz de Fora, MG – Brazil

[2]Research and Development Center for Communication Security (CEPESC)
Brasília, DF – Brazil.

beatriz.azevedo@estudante.ufjf.br, vinicius.lagrota@gmail.com,
moises.ribeiro@ufjf.br

Abstract. *This paper introduces a cost-effective function-selection procedure, anchored in Amdahl's Law, to orchestrate selective parallelization of Module-Lattice-Based Key-Encapsulation Mechanism Standard (ML-KEM) scheme on a dual-core ESP32 microcontroller. The procedure quantifies each function's single-core runtime and the overhead of parallel execution to isolate routines whose concurrent execution yields net time savings, while excluding low-latency tasks. The resulting ESP32 implementation eliminates separate single- and dual-core code paths, simplifies development, and achieves substantial reductions in overall runtime. Experimental evaluation on a dual-core ESP32 microcontroller of ML-KEM confirms the procedure's suitability for latency-critical applications.*

1. Introduction

The increasing connectivity among sensors and devices, driven by significant advances in the Internet of Things (IoT) technologies, has also heightened awareness of cyber-attack risks, making data protection a matter of national security[Nagrare et al. 2023, Jurcut et al. 2020]. In this context, cryptographic protocols play a crucial role in securing digital communications by ensuring the confidentiality, integrity, and authenticity of information [Stallings 2017]. Widely used schemes, such as Rivest-Shamir-Adleman (RSA) and Elliptic Curve Cryptography (ECC), rely on mathematical problems considered secure against conventional attacks. However, advances in quantum computing pose a threat to the security of these schemes, as sufficiently powerful quantum computers could break them in reduced time using techniques such as Shor's algorithm [Shor 1997]. Consequently, extensive research has been conducted to develop cryptographic schemes that remain secure even against adversaries equipped with such computers.

Recognizing the importance of these advancements, the National Institute for Standards and Technology (NIST) has been leading a standardization process for post-quantum cryptography (PQC) schemes, highlighting ML-KEM, standardized in FIPS-203 [NIST 2024b], as the primary key encapsulation algorithm due to its robustness and foundation on the Module-Learning With Errors (M-LWE) problem [Chen et al. 2016]. ML-KEM was designed to deal with embedded and resource-constrained microcontrollers, such as the ESP32 microcontroller; however, its implementation needs to be carefully designed to deal with latency-sensitive applications. There are a few works

(e.g., [Huang et al. 2024, Kim et al. 2024]), in which the Number Theoretic Transform (NTT) is designed to provide additional acceleration on polynomial multiplications used in ML-KEM. Considering low-cost platforms, [Junior and Henriques 2022, Ferro et al. 2021] demonstrated that performance optimizations in memory allocation and the selection of mathematical operations can enable PQC schemes, such as ML-KEM, to reduce computation time while maintaining efficiency and security. However, investigations into how to leverage multi-core microcontrollers, such as the ESP32, have not yet been conducted.

This study proposes the dual-core implementation of ML-KEM scheme, more specifically, ML-KEM-512, on an ESP32 microcontroller by proposing a cost-effective function selection procedure grounded in Amdahl's Law. In this procedure, we measure the single-core execution time of each function and quantify the overhead incurred by parallelization across both cores. By systematically comparing these values, the procedure isolates functions whose parallel execution yields net execution time gains. This procedure eliminates the need for separate single- and dual-core code paths, streamlining development and reducing engineering effort. Moreover, the procedure extends directly to ML-KEM-768 and ML-KEM-1024 (and other PQC schemes), providing a practical framework for optimizing latency-sensitive applications.

The remainder of this paper is organized as follows: Section 2 introduces the fundamental concepts; Section 3 details the function selection procedure; Section 4 discusses implementation details; Section 5 presents a comparative analysis; and Section 6 concludes with final remarks.

2. Fundamentals

With the advent of quantum computing, the security of classical cryptographic systems has become increasingly at risk. In 1994, Peter Shor developed an algorithm capable of factoring integers and solving the discrete logarithm problem in polynomial time, rendering classical cryptographic systems—widely used today—vulnerable in a quantum era, as they rely on the computational difficulty of mathematical problems [Shor 1994], such as integer factorization. Although large-scale quantum computers are not yet available, the threat has already materialized through the harvest now, decrypt later (HNDL) technique, which consists of an attack where adversaries intercept and store encrypted data transmitted over networks today, with the expectation that quantum computing will allow them to decrypt it in the future [Mosca 2018]. This type of attack demonstrates how information that is currently protected could be compromised in the future, affecting financial transactions, government data, and long-term classified information [Bernstein and Lange 2017].

In response to this reality, the NIST launched a standardization process for PQC algorithms in 2016, aiming to replace vulnerable cryptographic systems with secure alternatives that can withstand attacks from quantum computers [NIST 2016]. This process was divided into two main categories:

- **Key Encapsulation Mechanisms**: Cryptographic schemes that enable the secure exchange of symmetric keys between two parties.
- **Digital Signatures**: Algorithms that ensure the authenticity and integrity of digital documents.

After years of rigorous analysis and testing, the NIST announced in 2022 the selection of four algorithms for standardization, leading to the publication of FIPS-203, FIPS-204, FIPS-205, and FIPS-206, with the latter still in the documentation process [NIST 2024c, NIST 2024a, NIST 2024]. The CRYSTALS-Kyber (standardized as ML-KEM under FIPS-203 [NIST 2024b]) was chosen as the primary standard for secure key exchange, while the CRYSTALS-Dilithium, Falcon, and SPHINCS+ algorithms were selected as standards for digital signatures.

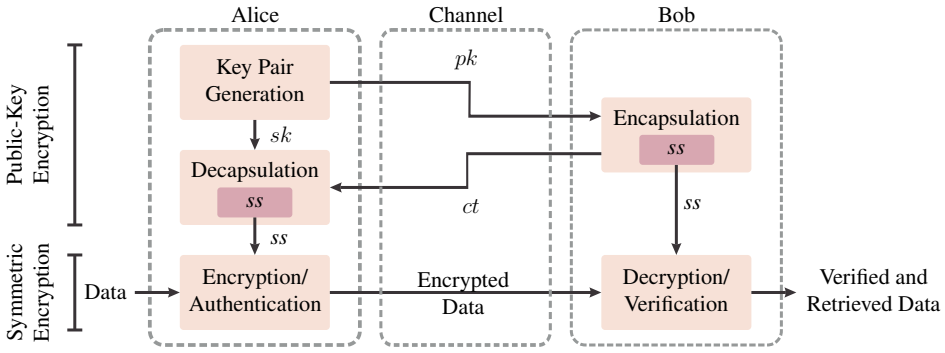
2.1. Key Encapsulation Mechanism

Key Encapsulation Mechanism (KEM) is a widely adopted solution to ensure confidentiality during key exchange. Its efficiency and simplicity easily allow the establishment of shared keys without the need for the explicit exchange of secrets between two parties. KEM is an asymmetric cryptographic scheme commonly used to securely exchange keys between two parties, making it particularly relevant in scenarios that require high security, such as post-quantum scenarios.

An illustration of the KEM process is shown in Figure 1. This process can be described in three main stages [Lagrota et al. 2022], as outlined below:

1. **Key Pair Generation**: This is the initial stage, in which Alice generates a key pair consisting of a public key pk and a private key sk. The public key is securely shared with Bob, while the private key remains securely stored with Alice.
2. **Encapsulation**: Using Alice's public key, Bob generates a random secret key ss and encapsulates it using the public key pk. The result of this process is a ciphertext ct, which is essentially the secret key encrypted with the public key pk.
3. **Decapsulation and Secret Key Retrieval**: Alice uses her private key sk to decapsulate the ciphertext ct, recovering the shared secret key ss. The mathematical primitive used must ensure that only the holder of the private key can access ss, even if ct is intercepted by third parties.

Figure 1. Illustration of the KEM process.



The secret key ss generated by a KEM enables the use of a symmetric cryptographic scheme for data exchange, as it is more efficient for this purpose. This is because symmetric cryptographic schemes, such as Advanced Encryption Standard (AES), offer superior performance in encrypting large volumes of data, while KEM ensures secure key distribution without transmitting the secret or key directly [Lagrota et al. 2022]. However, it is important to highlight that KEM, by itself, does not provide direct protection against

quantum computers. While it is an efficient mechanism for key encapsulation, its security depends on the underlying cryptographic primitive. Thus, it represents a viable solution for securing key exchange in the quantum era when combined with post-quantum cryptographic primitives, as these are specifically designed to withstand the aforementioned threats.

2.2. ML-KEM

ML-KEM represents an innovative advancement in the field of PQC, as its security foundation relies on the M-LWE problem. This structure employs modular lattices to ensure a high level of security, offering resistance against both classical and quantum computer attacks [NIST 2024d, Bos et al. 2018]. This mathematical choice reinforces the resilience of the cryptographic scheme and positions it as a viable solution in the face of future computational advancements, especially considering that potential vulnerabilities are more likely to emerge in schemes based on Ring-Learning With Errors (R-LWE) than in those based on M-LWE [Bos et al. 2021]. Unlike many post-quantum cryptographic schemes that impose heavy computational burdens, ML-KEM is specifically designed for efficient implementation on embedded and resource-constrained platforms. Additionally, the low cycle count and energy efficiency of ML-KEM make it ideal for battery-powered or energy-harvesting devices. These characteristics position ML-KEM as a strong candidate for deployment in the IoT and satellite communication.

Based on the Module-Learning With Errors (LWE) problem, ML-KEM offers lower computational complexity than other LWE schemes. In addition, to maximize its computational efficiency, ML-KEM integrates polynomial vectors with the NTT, optimizing critical operations such as modular multiplications and reducing complexity from $O(n^2)$ to $O(n \log n)$. ML-KEM offers flexible security levels with compact key and ciphertext sizes, corresponding to ML-KEM-512 (NIST Level 1), ML-KEM-768 (NIST Level 3), and ML-KEM-1024 (NIST Level 5). These levels are configured by setting the parameter `KYBER_K` to 2, 3, and 4, respectively, providing security comparable to AES-128, AES-192, and AES-256. This flexibility, combined with NIST standardization, allows the cryptographic scheme to be adapted to different security requirements without compromising its core structure.

Regarding its structure, ML-KEM follows the same three classic stages of a KEM: generation of a key pair (public and private)—represented by the `keygen` function; encapsulation of a secret key using the public key—represented by the `encap` function; and decapsulation of this secret key by the receiving party using its private key—represented by the `decap` function, as defined in the decapsulation step of ML-KEM specification [NIST 2016]. Additionally, the cryptographic scheme provides implementations that utilize SHAKE or AES-256 for pseudo-random generation of public keys, offering greater flexibility in configuring security parameters [Lagrota et al. 2022].

Another advantage related to ML-KEM is its implementations with constant-time execution and fixed-point arithmetic, which is essential for preventing timing attacks and ensuring compatibility with microcontrollers like the ARM Cortex-M4 [Kannwischer et al. 2019]. ML-KEM also features a compact memory footprint, requiring less than 4 kB of stack on common microcontrollers [Kannwischer et al. 2019, Kannwischer et al. 2020]. In the context of practical implementations, ML-KEM

demonstrates its potential in resource-constrained devices, such as ARM Cortex M4 [Abdulrahman et al. 2025] and ESP32 microcontroller [Segatz and Hafiz 2025].

Due to the hardware constraints of embedded devices—such as low power consumption, limited memory, and reduced processing capability—and the high computational cost of certain ML-KEM operations, it becomes necessary to identify critical points where parallelization techniques can be applied. In the case of the ESP32 microcontroller, it is worth mentioning that its dual-core architecture and *FreeRTOS* support provide advantages for implementing ML-KEM. However, the memory constraints (520 kB of SRAM) and clock limitations (up to 240 MHz) highlight the opportunity to leverage both cores to reduce the execution time of functions in ML-KEM, as detailed in Section 3.

2.2.1. ML-KEM Functions

For effectively leveraging the dual-core architecture of the ESP32 microcontroller requires a detailed understanding of the internal structure of ML-KEM to guide parallelization efforts strategically. In this context, its modular design of plays a fundamental role.

Figure 2 illustrates the modular organization of ML-KEM scheme, which is structured in layers that reflect both the functional decomposition of the algorithm and its potential for parallelization. At the top of the hierarchy, the KEM block encapsulates the full set of cryptographic operations and ensures indistinguishability under chosen ciphertext attack (IND-CCA) by introducing mechanisms such as ciphertext verification and re-encryption checks. These components are essential in real-world applications to ensure robustness against adaptive adversaries.

Directly beneath it lies the indistinguishability under chosen plain text attack (IND-CPA) block, which implements the core logic of key generation, encapsulation, and decapsulation. This layer is responsible for constructing the main cryptographic functionality of the scheme. It leverages structured polynomial operations to maintain confidentiality and relies heavily on data transformations built upon polynomial vectors.

Supporting the IND-CPA logic are the **Polynomial vectors** and **Polynomials** layers, which correspond to the `polyvec` and `poly` modules, respectively. These layers are tightly connected, as the `polyvec` module operates on vectors of polynomials, each of which is handled individually by the `poly` module. The `polyvec` layer is of particular interest in this work, as its structure naturally supports parallel execution: each polynomial in a vector can be processed independently, enabling effective distribution across cores. Typical operations at this level include applying the NTT, compression, modular multiplication, and vector addition. The independence of these computations means they can be parallelized with minimal synchronization overhead.

In contrast, although the `poly` module—represented in the **Polynomials** layer—is responsible for low-level operations on individual polynomials, its functions tend to be too fine-grained for parallelization to yield practical performance gains. Parallelizing at this level often results in excessive overhead from task management, negating any speedup. As such, parallelization efforts are best concentrated at the `polyvec` level.

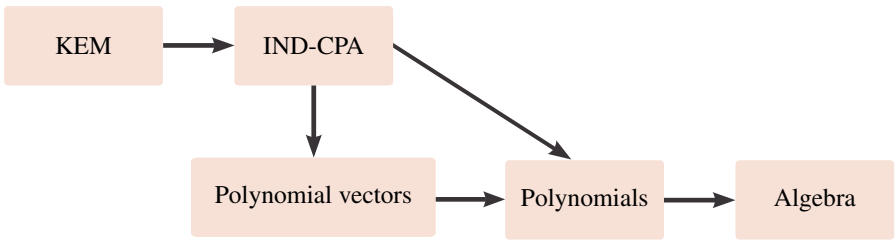
At the bottom of the hierarchy lies the **Algebra** block, which consists of the basic arithmetic operations that act on polynomial coefficients. These foundational routines

support the higher layers but are themselves not directly targeted for optimization in this study.

The modular decomposition, shown in Figure 2, guides the selective parallelization strategy adopted in this work. By focusing on layers where computational workloads are naturally independent—particularly within the `polyvec` module — it becomes possible to reduce execution time without incurring the penalties associated with unnecessary synchronization at lower abstraction levels.

Based on this modular decomposition, specific functions were selected for analysis according to their suitability for parallel execution. Functions from the `polyvec` module were prioritized because operations on individual polynomials are naturally independent, allowing effective distribution of tasks across cores with minimal synchronization overhead. Functions from the `IND-CPA` module were selectively included where it was possible to parallelize isolated segments, such as noise generation, without introducing correctness risks. Additionally, functions from the `poly` module were evaluated to demonstrate that parallelization at too fine a granularity, such as the individual polynomial level, often leads to excessive task management overhead and performance degradation. This selection strategy ensures that the study focuses on parallelization opportunities that are both practical and impactful within the dual-core environment.

Figure 2. Illustration of ML-KEM.



Among the functions in the `polyvec` module, the following are noteworthy:

- `polyvec_ntt` and `polyvec_invntt_tomont`: apply the NTT and its inverse, respectively, to all polynomials in the vector.
- `polyvec_basemul_acc_montgomery`: multiplies and accumulates the coefficients of the polynomials stored in two `polyvec` vectors. This function is essential for key generation and encapsulation.
- `polyvec_compress` and `polyvec_decompress`: handle the compression and decompression of polynomial vectors.
- `polyvec_reduce`: performs modular reduction of polynomial coefficients to ensure they remain within the valid range for subsequent operations.
- `polyvec_add`: computes the element-wise addition of two polynomial vectors.
- `polyvec_tobytes` and `polyvec_frombytes`: convert polynomial vectors to and from their binary representations, ensuring compatibility with storage and transmission formats.

In addition to the functions in the `polyvec` module, functions from the `IND-CPA` and `poly` modules were also analyzed. In the `IND-CPA` module, `gen_matrix`, `indcpa_keypair_derand`, and `indcpa_enc` were selected, with only parallelizable segments modified in the last two. This is because other functions

in `IND-CPA` exhibit strong sequential dependencies or frequent shared data access, making them unsuitable for parallel processing. Conversely, the analysis of the `poly` module was conducted to show that parallelizing at lower levels introduces significant overhead, ultimately degrading performance rather than improving it. As such, the following functions from `IND-CPA` were selected:

- `gen_matrix`: generates the public matrix using pseudorandom coefficients, with independent processing for each entry.
- `indcpa_enc`: performs key encapsulation. To avoid dependency issues, only noise generation (`indcpa_noise_eta`) was tested.
- `indcpa_keypair_derand`: optimizes noise generation in key pair creation. Similar to `indcpa_enc`, only its noise generation function (`indcpa_noise_keypair`) was parallelized.

In the `poly` module, the `poly_tomont` function was selected for analysis due to its role in converting polynomial coefficients to the Montgomery representation, which is essential for modular arithmetic operations and for reducing the protocol's execution time.

3. Function Selection Procedure

In embedded devices, it is crucial to adopt strategies that reduce computational cost without compromising the effectiveness of the cryptographic scheme. In the case of ML-KEM, such strategies are built upon three main pillars: (i) restructuring the code to eliminate unnecessary operations (e.g., having a different NTT that demands less computational complexity, which can be efficiently performed on an ESP32 microcontroller; (ii) leveraging specialized processor instructions, commonly available in architectures like x86 (AVX) and ARM (Neon), that support single instruction, multiple data (SIMD) operations, which are not available on the ESP32 microcontroller; and (iii) implementing a *multicore* execution model that distributes the workload across available processing cores, which is feasible given the ESP32's dual-core architecture.

Considering the third pillar, this section details a selection procedure for the efficient parallelization of functions within ML-KEM on a dual-core ESP32 microcontroller. We also introduce a criterion that allow us select the functions offering advantages when they are parallelized. In Subsection 3.1, we detail the flow-chart of the function selection procedure to show the benefits of parallelizing functions of ML-KEM for reducing their execution time. In the sequel, Subsection 3.2 shows how to obtain a threshold time that for performing a decision-making in favor or not of parallelization of functions in a dual-core ESP32 microcontroller.

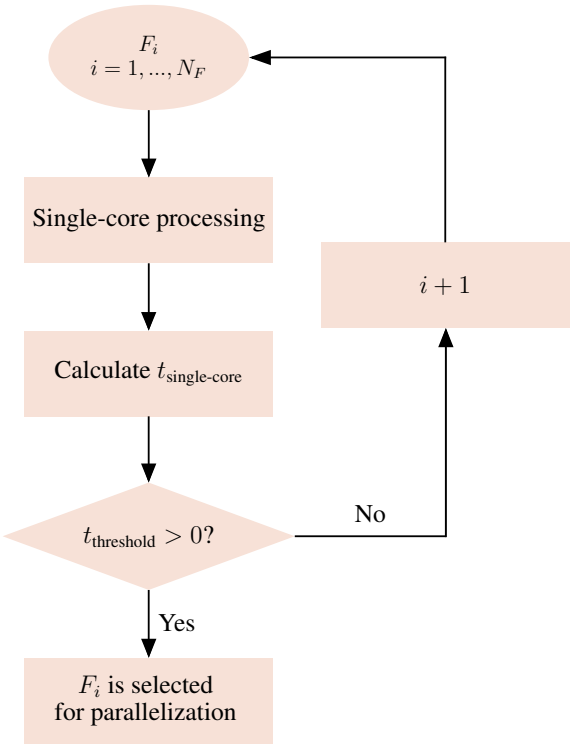
3.1. Flow Chart

To effectively leverage the dual-core architecture of the ESP32 microcontroller, we propose a procedure based on a simple yet effective criterion to identify which ML-KEM functions can attain less execution time when its processing is parallelized.

Figure 3 shows the flow-chart for this procedure. In this model, we assume that there are F_i, $i = 1, \cdots, N_F$ functions in ML-KEM. According to this flow-chart, the ith function is solely executed in a single-core of the ESP32 microcontroller and its execution

time ($t_{\text{single-core}}$) is obtained. In the following, the execution time $t_{\text{single-core}}$ is compared to a quantity to generate the so-called threshold time $t_{\text{threshold}}$. This threshold time represents the minimum single-core execution time required for a function to benefit from parallelization. It is derived by accounting for the overhead introduced by task management and synchronization, and is further discussed in Section 3.2. If $t_{\text{threshold}} > 0$, then the ith function is marked as suitable for parallelization, and its execution across both cores is advantageous for reducing the total execution time. Otherwise, the ith function remains assigned to a single-core execution, thereby avoiding the performance degradation by unnecessary parallelizing functions. Overall, this procedure serves not only as a practical guideline for implementing efficient parallelism but also as a safeguard against applying parallelization strategies that, despite being theoretically promising, fail to deliver benefits in resource-constrained or real-time execution scenarios.

Figure 3. The procedure flow-chart



3.2. Threshold Time ($t_{\text{threshold}}$)

Evaluating the benefits of parallelizing a task requires defining a threshold execution time that justifies the overhead incurred when managing *tasks* in a dual-core implementation. In this context, the threshold time ($t_{\text{threshold}}$) at which parallelization becomes advantageous depends on two key parameters: (i) the execution time of a task on a single core, denoted by $t_{\text{single-core}}$, and (ii) the overhead time required for creating, synchronizing, and terminating *tasks* in a dual-core setup, denoted by t_{overhead}.

While Amdahl's Law informs the ideal speedup of a *multicore* implementation under the assumption of zero overhead time [Hill and Marty 2008], we adapt it to include the overhead time as follows:

$$S \triangleq \frac{1}{(1-P) + \frac{P}{N} + \epsilon}, \tag{1}$$

in which S is the speedup, $P \in \mathbb{R} | \, 0 \leq P \leq 1$ is the fraction of the task that can be parallelized, $N \in \mathbb{N}_+$ is the number of cores, and $\epsilon \in \mathbb{R}_+$ denotes the fraction of execution time consumed by the overhead associated with using N cores. Assuming $N = 2$, $P = 1$, and $\epsilon = \frac{1}{2}$, the resulting speedup is $S = 1$, indicating that parallelization yields no performance gain. Thus, knowing the overhead time t_{overhead} is essential for determining a threshold time $t_{\text{threshold}}$ for selecting a task for the execution in parallel.

Now, let us consider a task from ML-KEM whose execution time on a single core is $t_{\text{single-core}}$. The execution time on a dual-core is then given by

$$t_{\text{dual-core}} = \frac{t_{\text{single-core}}}{2} + t_{\text{overhead}}, \tag{2}$$

where the term $\frac{t_{\text{single-core}}}{2}$ represents the parallel execution of two halves of the task (i.e., $P = 1$), and t_{overhead} is the overhead for using two cores. Based on these quantities, we can also obtain the speedup using the following expression:

$$S = \frac{1}{1-\beta}, \tag{3}$$

in which

$$\beta = \frac{t_{\text{single-core}} - t_{\text{dual-core}}}{t_{\text{single-core}}} \tag{4}$$

is named execution time gain. A positive β indicates a reduction in execution time, while $\beta \leq 0$ indicates no gain or a slowdown.

Figure 4 illustrates the relationship among $t_{\text{single-core}}$, $t_{\text{dual-core}}$, and t_{overhead}. For parallelization to be beneficial, the following condition must be satisfied:

$$t_{\text{single-core}} \geq t_{\text{dual-core}}. \tag{5}$$

Substituting (2) into (5) yields

$$t_{\text{single-core}} \geq 2t_{\text{overhead}}. \tag{6}$$

This inequality leads to the definition of the threshold time, expressed as

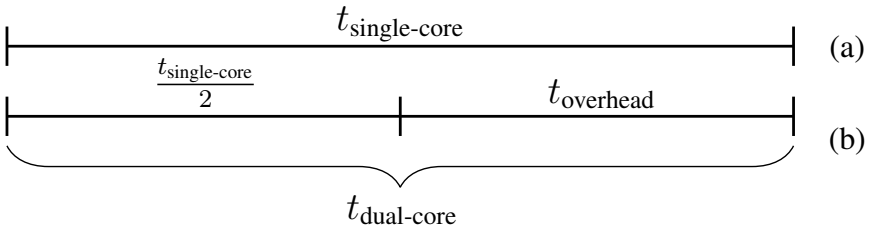
$$t_{\text{threshold}} \triangleq t_{\text{single-core}} - f(t_{\text{overhead}}), \tag{7}$$

where $f(t_{\text{overhead}}) = 2t_{\text{overhead}} + \gamma$, and $\gamma \in \mathbb{R}_+$ is an optional offset that accounts for additional design considerations (e.g., development time for implementing parallelization). In this study, we assume $\gamma = 0$. Thus, if $t_{\text{threshold}} > 0$, parallelizing the task is advantageous; otherwise, if $t_{\text{threshold}} \leq 0$, parallelization does not provide execution time benefits.

Based on experimental results, we observe that the speedup (3), execution time gain (4), and threshold time (7) can all be used to identify functions that benefit from

Figure 4. Illustration of the relationship between $t_{\text{single-core}}$, $t_{\text{dual-core}}$, and t_{overhead}.

(a) Execution time on a single-core. (b) Execution time on dual-core.



parallelization. However, it is important to emphasize that the threshold time is significantly more cost-effective because it requires knowledge only of $t_{\text{single-core}}$ and $f(t_{\text{overhead}})$, whereas the other metrics require both $t_{\text{single-core}}$ and $t_{\text{dual-core}}$. Obtaining $t_{\text{dual-core}}$ for each function is considerably more complex, as it requires developing code for both dual-core and single-core execution modes.

It is important to analyze how the parameter `KYBER_K` influences the speedup. When `KYBER_K` is even—enabling a balanced distribution of tasks on a dual-core (i.e., $P = 1$ and $N = 1$)—a speedup approaching $S = 2$ is expected if $t_{\text{single-core}} \gg 2t_{\text{overhead}}$ (i.e., $\epsilon \to 0$, meaning the impact of the parallelization overhead becomes negligible. However, when `KYBER_K` is odd, one core must process a portion of the task sequentially, which prevents full parallelization and results in a lower speedup. For instance, in the specific case of `KYBER_K` $= 3$, two of the three operation sets can be executed in parallel, while the third must be processed separately, resulting in $P = \frac{1}{2}$. Consequently, the speedup tends to $S = \frac{4}{3}$ if $t_{\text{single-core}} \gg 2t_{\text{overhead}}$ (i.e., $\epsilon \to 0$). One can note that `KYBER_K` $= 4$, we consider $N = 2$ and $P = 1$, and, consequently, the speedup approximates $S = 2$ when $t_{\text{single-core}} \gg 2t_{\text{overhead}}$ (i.e., $\epsilon \to 0$).

In general, the threshold time $t_{\text{threshold}}$ serves as a useful parameter for evaluating the effectiveness of task parallelization when accounting for the overhead associated with task creation and synchronization.

3.2.1. Threshold Time Value

The determination of t_{overhead} requires precise measurement of execution time, as synchronization latency between processor cores can vary depending on the architecture, operating system, and other factors specific to the execution environment. To estimate this value, we conducted an experiment in which two empty *tasks* were created and assigned to different cores of the ESP32 microcontroller. This setup allows the exclusive measurement of synchronization cost. The execution time was recorded from the initiation to the completion of both tasks, yielding a value of $t_{\text{overhead}} = 68\ \mu s$. Consequently, the minimum threshold for a function to benefit from parallel execution on the ESP32 is given by

$$t_{\text{threshold}} = t_{\text{single-core}} - 136\ \mu\text{s}. \tag{8}$$

Note that (8) enables evaluation and prediction of the system's behavior when distributing *tasks* across dual cores. However, in practice, execution time is also affected by other variables, such as inter-task interference and hardware-specific synchronization overheads.

To further validate this approach, an additional experiment was conducted using the `polyvec_add` function, which was selected due to its short runtime ($t_{\text{single-core}} \approx 47.3\ \mu$s) and tunable computational complexity. This function is employed in the case of `KYBER_K` $= 2$, for which we assume $P = 1$. The experiment consisted of repeating the `polyvec_add` function $n = 1, \ldots, 30$ times in both single-core and dual-core configurations and measuring their corresponding execution times, denoted $t_{\text{single-core}}$ and $t_{\text{dual-core}}$, respectively. The execution time gain was then computed by

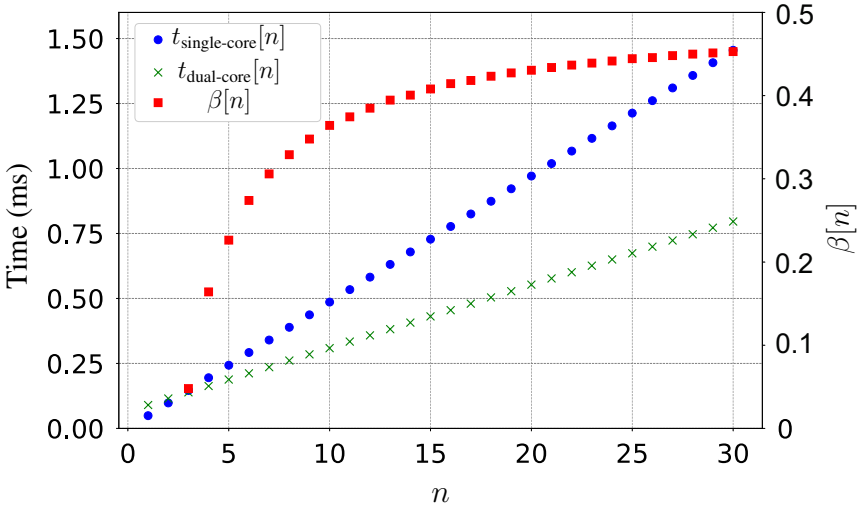
$$\beta[n] = \frac{\Delta t_{\text{single-core}}[n] - \Delta t_{\text{dual-core}}[n]}{\Delta t_{\text{single-core}}[n]}, \tag{9}$$

where $\Delta t_{\text{single-core}}[n]$ denotes the execution time of the nth repetition in the single-core setting, and $\Delta t_{\text{dual-core}}[n]$ refers to the corresponding time in the dual-core setting. A value of $\beta[n] > 0$ indicates a performance gain through parallelization, while $\beta[n] \leq 0$ implies no gain or a performance degradation.

Figure 5 presents the curves of $\Delta t_{\text{single-core}}[n]$, $\Delta t_{\text{dual-core}}[n]$, and $\beta[n]$. The intersection of $\Delta t_{\text{single-core}}[n]$ and $\Delta t_{\text{dual-core}}[n]$ occurs around $\hat{t}_{\text{threshold}} \approx 142\ \mu$s, identifying the point beyond which parallel execution becomes advantageous. The difference $t_{\text{threshold}} - \hat{t}_{\text{threshold}} = -6\ \mu$s does not indicate a real discrepancy but is attributed to the measurement precision limitation for the `polyvec_add` function, whose execution time increases with computational complexity. Nonetheless, this experiment supports the appropriateness of considering $t_{\text{threshold}} = 136\ \mu$s.

Finally, this experiment demonstrates that the execution time gain $\beta[n]$ asymptotically approaches a value close to 0.5 as $n \to \infty$. Thus, the parallel speedup can be expressed as $S = \lim_{n\to\infty} \frac{1}{1-\beta[n]}$. Given that $\lim_{n\to\infty} \beta[n] \approx 0.47$, and assuming $N = 2$ and $P = 1$, manipulation of (1) yields $\epsilon = 0.03$.

Figure 5. Execution time analysis of the `polyvec_add` function for `KYBER_K` $= 2$.



4. Implementation Details

This section presents the architectural and system-level strategies adopted in the parallel implementation of ML-KEM scheme on the dual-core ESP32 platform. The design aimed

to minimize overhead, promote deterministic execution, and maintain reproducibility within the limitations of embedded systems. The implementation was built on *FreeRTOS*, the real-time operating system natively supported by the ESP32. To ensure predictable performance and avoid the latency associated with dynamic memory management, all parallel tasks are allocated statically at system startup.

For each function targeted for parallelization, such as `polyvec_ntt` and `polyvec_basemul_acc_montgomery`, two tasks were instantiated and permanently pinned to core 0 and core 1, respectively. These tasks are reused across all parallel operations to further reduce runtime overhead. To enable concurrent processing with minimal synchronization, we partition the data structures—primarily polynomial vectors—evenly between the two cores. This approach allows each task to operate independently on disjoint memory regions, thereby eliminating the need for mutual exclusion mechanisms.

The synchronization is handled using binary semaphores, which offer a lower overhead for signaling between independent cores. A central control task orchestrates the execution flow: it first signals both worker tasks to begin a parallel operation and then waits to receive a completion signal from each. This protocol guarantees that both workers have finished before the main program continues, ensuring algorithmic correctness.

The decision to use semaphores rather than `mutexes` was intentional. While mutexes are appropriate for protecting shared resources, semaphores offer lower overhead and are better suited for signaling across independent processing units. In this context, semaphores provided an efficient and reliable means of coordinating task execution between the two cores.

5. Comparative Analysis

Aiming to validate the effectiveness of the proposed function selection criterion, this section discusses the experimental results obtained from the implementation of ML-KEM-512 (`KYBER_K` $= 2$) on the ESP32 microcontroller when non-selective and the proposed function selection procedures are considered. All performance metrics reported in this section were obtained through 101 repeated executions of each operation on the ESP32 platform running *FreeRTOS*. The median execution time was adopted to mitigate the effects of sporadic variations in task scheduling and clock drift. In this sense, Subsection 5.1 shows how the function selection procedure succeed to take advantage of dual-core. In the sequel, Subsection 5.2 focuses the gains obtained when the proposed function selection procedure is applied. Our implementation can be found in [Azevedo and Lagrota 2025].

5.1. Using the Function Selection Procedure

This subsection evaluates the execution time of the functions from Section 2.2.1 for the case `KYBER_K` $= 2$ in ML-KEM scheme, implemented both on a single-core (without parallelization) and on dual-core (with parallelization). Table 1 summarizes, in milliseconds, the values of $t_{\text{single-core}}$, $t_{\text{dual-core}}$, $t_{\text{threshold}}$, corresponding execution time gain β and speedup value S for each function.

We can see that functions with index < 6 (i.e., $t_{\text{threshold}} < 0$) show no benefit from parallelization, which is confirmed by $\beta \leq 0$, and, consequently, $S \leq 0$. These include `polyvec_from_bytes`, `polyvec_add`, `polyvec_to_bytes`,

Table 1. Performance comparison: wo/ and w/ parallelization.

Index	Function	$t_{\text{single-core}}$	$t_{\text{dual-core}}$	$t_{\text{threshold}}$	β	S
1	`polyvec_frombytes`	0.048	0.106	< 0	-1.208	0.453
2	`polyvec_add`	0.049	0.106	< 0	-1.163	0.462
3	`polyvec_tobytes`	0.073	0.113	< 0	-0.548	0.646
4	`poly_tomont`	0.065	0.097	< 0	-0.492	0.670
5	`polyvec_decompress`	0.103	0.106	< 0	-0.029	0.972
6	`polyvec_reduce`	0.120	0.120	$= 0$	0.000	1.000
7	`polyvec_compress`	0.179	0.156	> 0	0.128	1.147
8	`polyvec_basemul_acc_montgomery`	0.448	0.341	> 0	0.239	1.314
9	`indcpa_noise_eta`	1.074	0.638	> 0	0.406	1.684
10	`polyvec_ntt`	0.802	0.471	> 0	0.413	1.704
11	`indcpa_noise_keypair`	1.363	0.787	> 0	0.423	1.733
12	`gen_matrix`	2.049	1.163	> 0	0.432	1.761
13	`polyvec_invntt_tomont`	1.210	0.666	> 0	0.450	1.818

`poly_tomont`, `polyvec_decompress`, and `polyvec_reduce`. For example, `polyvec_add` grows from 0.049 ms (single-core) to 0.106 ms (dual-core), and `polyvec_decompress` from 0.103 ms to 0.106 ms—showing that synchronization overhead dominates any parallel gain.

In contrast, functions with index > 6 (i.e., $t_{\text{threshold}} > 0$)—those exceeding 136 μs on single-core—do benefit from dual-core execution. Specifically, `polyvec_compress`, `polyvec_basemul_acc_montgomery`, `indcpa_noise_eta`, `polyvec_ntt`, `indcpa_noise_keypair`, `gen_matrix`, and `polyvec_invntt_tomont` exhibit $\beta > 0$, and, consequently $S > 1$. Here, the execution time is sufficient to amortize overhead time, yielding clear performance improvements. Moreover, as $t_{\text{single-core}} \gg 2\,t_{\text{overhead}}$, we observe $\beta \to 0.5$, in agreement with the theoretical speedup, which is obtained from (1) when $N = 2$, $P = 1$, and $\epsilon \to 0$.

In summary, only functions whose single-core runtime exceeds the time threshold ($t_{\text{threshold}} > 0$) achieve meaningful acceleration under dual-core execution. However, setting $\gamma \neq 0$ can prevent the parallelization of functions that offer only marginal execution time gains, thereby reducing the overall effort required to develop parallelized code.

5.2. Performance Validation

To validate the effectiveness of the proposed function selection procedure, we perform a performance comparison between the three main routines of the Application Programming Interface (API) of ML-KEM—`keygen`, `encap`, and `decap`—responsible for key pair generation, encapsulation, and decapsulation, respectively. These routines are among the most computationally intensive operations in ML-KEM protocol, as they heavily rely on a specific set of functions detailed in Section 2.2.1. Since the functions within this set are central to the overall performance of the algorithm, their behavior has a profound impact on the execution time of the entire system. Furthermore, the routines `keygen`, `encap`, and `decap` are fundamental to ML-KEM workflow, and their performance directly influences the efficiency of the cryptographic operations. In this sense, the following implementations of these routines are considered:

1. **Implementation Reference (Impl. Ref.)**: all functions in the routines are executed on a single-core, serving as the reference for performance comparison. The

execution time is denoted by $t_{\text{Impl. Ref.}}$.

2. **Implementation** $\#2$ **(Impl.** $\#1$**)**: all functions in the routines are executed on a dual-core, allowing us to evaluate the effects of indiscriminate parallelization. The execution time is denoted by $t_{\text{Impl. }\#1}$.
3. **Implementation** $\#3$ **(Impl.** $\#2$**)**: refers to the use of the proposed function selection procedure to decide the functions in the routine that are executed on a dual-core, while the remaining functions are executed on a single-core. According to Table 1 in Subsection 5.1, the functions with indexes between 1 and 6 are executed on single-core, while the functions with indexes between 7 and 13 are executed on dual-core. The execution time is denoted by $t_{\text{Impl. }\#2}$.

Table 2 presents the execution times of `keygen`, `encap`, and `decap` routines, denoted by t_{keygen}, t_{encap}, and t_{decap}, respectively. It also includes the total execution time for each implementation, which is given by

$$t_{\text{total}} = t_{\text{keygen}} + t_{\text{encap}} + t_{\text{decap}}. \tag{10}$$

Performance is compared using the execution time gain for these routines, which is adapted from (4) and given by

$$\beta_{\#i} = \frac{t_{\text{Impl. Ref.}} - t_{\text{Impl. }\#i}}{t_{\text{Impl. Ref.}}}. \tag{11}$$

where, $i \in \{1, 2\}$. Moreover, we calculate the resulting speedup as follows:

$$S_{\#i} = \frac{1}{1 - \beta_{\#i}}. \tag{12}$$

All execution times in Table 2 are given in milliseconds. **Impl.** $\#1$ achieves a substantial speedup over **Impl. Ref.**, as evidenced by $S_{\#1}$. **Impl.** $\#2$ delivers a further, albeit smaller, gain relative to **Impl.** $\#1$, as indicated by $S_{\#2} \gtrsim S_{\#1}$. Overall, the results in Table 2 validate the proposed Function Selection Procedure and underscore the importance of selecting the appropriate functions for parallelization in the routines aiming to minimize the total execution time of ML-KEM scheme. Moreover, they emphasize that experimental evaluation on the target multi-core (i.e., EXP32 microcontroller) is mandatory, as the choice of which functions to parallelize has a significant impact on performance improvements.

Table 2. Performance comparison between the three implementations.

Description	**Impl. Ref.**	**Impl.** $\#1$	**Impl.** $\#2$	$\beta_{\#1}$	$\beta_{\#2}$	$S_{\#1}$	$S_{\#2}$
t_{keygen}	7.460	5.273	5.141	0.293	0.311	1.414	1.451
t_{encap}	8.990	6.468	6.352	0.281	0.293	1.391	1.414
t_{decap}	11.214	8.240	8.120	0.265	0.276	1.361	1.381
t_{total}	27.665	19.993	19.621	0.277	0.291	1.383	1.410

6. Conclusion

This work focused on a dual-core implementation of ML-KEM-512 scheme (KYBER_K = 2) on an ESP32 microcontroller, driven by a cost-effective function selection procedure based on single-core execution time and parallelization overhead. By eliminating the need for distinct single- and dual-core code, our proposal streamlines development and reduces engineering effort. We also highlighted how Amdahl's Law can guide this investigation. Experimental results on a dual-core ESP32 microcontroller confirm that the procedure effectively identifies functions that benefit most from parallel execution, yielding substantial latency reductions, which can be useful for IoT devices.

Overall, the proposed function selection procedure cost-effectively ensures that the benefits of parallelization outweigh the overhead associated with task management and synchronization related to parallelization.

Although the approach proved effective for the target use case, it is important to acknowledge its limitations. In cases where data dependencies are stronger or inter-core coordination becomes more complex, the model may not fully capture the real performance dynamics, and further prototyping would be necessary. The proposed methodology is best suited for functions that are trivially parallelizable and operate over independent memory segments. Future work could involve adapting this strategy for more complex cryptographic schemes or testing it on heterogeneous hardware architectures.

Acknowledgments

This research was supported in part by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) under Grant 001, Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) under grants 445958/2024-3 and 314741/2020-8, and Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG) under grant APQ-04623-22.

References

Abdulrahman, A., Kannwischer, M. J., and Lim, T.-H. (2025). Enabling microarchitectural agility: Taking ML-KEM & ML-DSA from cortex-m4 to m7 with SLOTHY. Cryptology ePrint Archive, Paper 2025/366.

Azevedo, B. and Lagrota, V. (2025). Dual core ML-KEM on ESP32. `https://github.com/beatrizufjf/Dual-core-ML-KEM-on-ESP32.git`.

Bernstein, D. J. and Lange, T. (2017). Post-quantum cryptography—dealing with the fallout of physics success. Cryptology ePrint Archive, Paper 2017/314.

Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J. M., Schwabe, P., Seiler, G., and Stehle, D. (2018). Crystals - kyber: A cca-secure module-lattice-based kem. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 353–367.

Bos, J. W., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J., Schwabe, P., Stehlé, D., and Tibouchi, M. (2021). CRYSTALS-Kyber Algorithm Specifications and Supporting Documentation (Round 3). `https://pq-crystals.org/kyber/data/kyber-specification-round3-20210131.pdf`. Accessed April 2025.

Chen, L., Jordan, S., Liu, Y.-K., Moody, D., Peralta, R., Perlner, R., and Smith-Tone, D. (2016). Report on Post-Quantum cryptography. Technical report, National Institute of Standards and Technology.

Ferro, L. F. C., Rampazzo, F. J. A., and Henriques, M. A. A. (2021). Estudos de otimização do algoritmo de criptografia pós-quântica crystals-kyber. In *Anais do XVIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSEG)*.

Hill, M. D. and Marty, M. R. (2008). Amdahl's law in the multicore era. *Computer*, 41(7):33–38.

Huang, J., Zhao, H., Zhang, J., Dai, W., Zhou, L., Cheung, R. C. C., Koç, c. K., and Chen, D. (2024). Yet another improvement of plantard arithmetic for faster kyber on low-end 32-bit iot devices. *IEEE Transactions on Information Forensics and Security*, 19:3800–3813.

Junior, G. G. and Henriques, M. A. A. (2022). Redução do consumo de memória no algoritmo de criptografia pós-quântica saber em microcontroladores arm cortex-m0+. In *Seminários em Engenharia de Computação e Sistemas Digitais*.

Jurcut, A., Niculcea, T., Ranaweera, P., and Le-Khac, N.-A. (2020). Security considerations for internet of things: A survey. *SN Comput. Sci.*, 1(4).

Kannwischer, M. J., Rijneveld, J., Schwabe, P., and Stoffelen, K. (2019). Efficient masking of Kyber on ARM cortex-m4. In *Proceedings of the 12th Workshop on Embedded Systems Security (WESS)*. ACM.

Kannwischer, M. J., Schwabe, P., and Stoffelen, K. (2020). pqm4: Testing and benchmarking nist pqc on the arm cortex-m4. In *Proceedings of CARDIS 2020: Smart Card Research and Advanced Applications*. Springer.

Kim, Y., Yoon, S., and Seo, S. C. (2024). Vectorized implementation of kyber and dilithium on 32-bit cortex-a series. *IEEE Access*, 12:104414–104428.

Lagrota, V., Camponogara, Â., López, J., and Ribeiro, M. V. (2022). The feasibility of the crystals-kyber scheme for smart metering systems. *IEEE Access*, 10:131303–131317.

Mosca, M. (2018). Cybersecurity in an era with quantum computers: Will we be ready? *IEEE Security & Privacy*, 16(5):38–41.

Nagrare, T., Sindhwad, P., and Kazi, F. (2023). BLE protocol in IoT devices and smart wearable devices: Security and privacy threats.

NIST (2016). Submission requirements and evaluation criteria for the post-quantum cryptography standardization process. `https://csrc.NIST.gov/projects/post-quantum-cryptography`. Accessed: 2024-12-30.

NIST (2024). Stateless hash-based digital signature standard. Technical report, Washington, D.C.

NIST, G. M. D. (2024a). Module-Lattice-Based digital signature standard. Technical report, Gaithersburg, MD.

NIST, G. M. D. (2024b). Module-lattice-based key-encapsulation mechanism standard. Technical report, Gaithersburg, MD.

NIST, G. M. D. (2024c). Module-lattice-based key-encapsulation mechanism standard. Technical report, Gaithersburg, MD.

NIST, G. M. D. (2024d). Module-lattice-based key-encapsulation mechanism standard. Technical report, Gaithersburg, MD.

Segatz, F. and Hafiz, M. I. A. (2025). Efficient implementation of CRYSTALS-KYBER key encapsulation mechanism on ESP32.

Shor, P. W. (1994). Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 124–134, Los Alamitos, CA. IEEE.

Shor, P. W. (1997). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509.

Stallings, W. (2017). *Cryptography and Network Security: Principles and Practice*. Pearson, Upper Saddle River, NJ, 7 edition.

Tailoring a Genetic Algorithm for searching Shortest Lattice Vector in NTRU Lattices

Raul Caram de Assis [1], Thiago do Rêgo Sousa[2]

[1 2] Centro de Pesquisa e Desenvolvimento para a Segurança das Comunicações (Cepesc)

raulcaram21@gmail.com, thiagodoregosousa@gmail.com

Abstract. *The NTRU cryptosystem relies on the hardness of the Shortest Vector Problem (SVP), a central challenge in lattice-based cryptography. While genetic algorithms (GAs) have shown promise in solving SVP on generic lattices, little attention has been given to applying it to cryptographic structures like NTRU. In this work, we investigate the suitability of a GA specifically tailored to NTRU lattices. Our approach involves searching outside the lattice and analyzing a cost function whose behavior appears smoother when using BKZ-reduced bases. We present initial observations and outline directions for future research.*

1. Introduction

The NTRU system, proposed in [Hoffstein et al. 1998], is one of the most popular post-quantum cryptosystems whose security depends on the difficulty of solving a shortest vector problem (SVP) which is considered a classical lattice problem for which no efficient quantum or classical algorithm has been developed so far for the general case.

Many different methods have been developed to solve, approximately or exactly, the SVP and another highly related problem: the closest vector problem (CVP). Most notably, basis reduction algorithms LLL and BKZ and their variants, as well as sieving, prunning, random sampling and a series of different stochastic-search based algorithms or a combination of those.

Genetic algorithms (GAs) have proven effective in navigating large combinatorial search spaces and finding near-optimal solutions to hard problems, including the SVP. The starting point for a successful application of GAs to solve SVP was in the article [Ding et al. 2015] in which the authors improved upon some former solutions submitted to the Darmstadt challenges (see [Darmstadt lattice challenge]).

In the context of GAs, one of the first works was [Ding and Zhu 2014] that proposed a random sampling method based on y-sparse representations. [Ding et al. 2015] introduced a coordinate system based on the Gram-Schmidt orthogonalized basis along with binary encoding. [Luan et al. 2020] extended this idea using the Natural Number Representation introduced by [Fukase and Kashiwabara 2015]. [Reddy and Rao 2021] and [Fukase and Kaminaga 2022] further improved using evolutionary approaches. More recently, [Moghissi and Payandeh 2019] explored parallelization for SVP using evolutionary strategies.

Despite these advances, most research has focused on generic lattice instances from SVP challenge sets, with limited attention to cryptographic systems like NTRU. In this work, we investigate the potential of tailoring genetic algorithms specifically to attack the NTRU cryptosystem. Rather than applying general-purpose GA techniques, we

propose a domain-specific adaptation and examine its effectiveness. This paper presents preliminary ideas, outlines the design of our customized algorithm, and identifies open research questions that must be addressed to assess the practicality and scalability of such approaches in cryptanalytic contexts. The rest of the paper is organized as follows: In Section 2 we describe the NTRU system, the SVP problem that is associated to recovering its private key and the whole of genetic algorithms in solving SVP for general lattices. Section 3 presents our idea to tailor GAs to NTRU by searching for points outside the lattice and using an appropriate cost function. Section 4 concludes and gives further research directions.

2. SVP, NTRU and Genetic algorithms

2.1. Lattices and SVP

Given a basis $B = (\vec{b}_0, \ldots, \vec{b}_{n-1})$ containing linearly independent vectors in $\mathbb{R}^n$, to be treated as the $n \times n$ matrix having $\vec{b_i}$ as its $(i+1)$-th row; we define $L(B) = \{\vec{x}B | \vec{x} \in \mathbb{Z}^n\}$ as the lattice generated by B, and if B is clear from the context or not relevant we may simply write it as L. This is the set of all points in $\mathbb{R}^n$ that can be reached by starting from the origin $\vec{0}$ and through finitely many operations of adding or subtracting vectors in B. The basis is never unique and in fact for every lattice of dimension $n \geq 2$ there is an infinite number of different basis. The SVP for L is the problem of finding one $\vec{v} \in L^* = L \setminus \vec{0}$ such that for any $\vec{w} \in L^*$, $\|\vec{v}\| \leq \|\vec{w}\|$.

2.2. NTRU cryptosystem

NTRU is an asymmetric cryptographic system designed by [Hoffstein et al. 1998] to be quantum-safe. The system is based upon operations in a given convolutional polynomial ring $R_q = (\mathbb{Z}[x]/q\mathbb{Z})/(x^N - 1)$, for given $q, N \in \mathbb{N}_{>0}$ co-primes, with a pair of operations $(+, *)$ defined. In the context of NTRU, the reduction modulo q is 'center-lifted', *i.e.* it takes any integer z and returns the unique $z' \in C_q := \mathbb{Z} \cap (-q/2, q/2]$ such that $q|(z - z')$.

The polynomials in R_q have degree at most $N-1$ and its coefficients are members of C_q. The operations $+$ and $*$ (sum and convolution, respectively) of two polynomials in R_q are to be done as the ordinary sum/product of two polynomials followed by the application of two reductions: the integer coefficients are reduced modulo q and the exponents of the variables $x^0, x^1, \ldots$ are reduced modulo N.

In NTRU the private keys $\vec{f} = (f_0, \ldots, f_{N-1})$ and $\vec{g} = (g_0, \ldots, g_{N-1})$ are vectors with entries in $\{-1, 0, 1\}$ (more precisely they have a non-zero, fixed or random, quantity of each of these possible values). The parameters $\vec{f}$ and $\vec{g}$ are generated independently and from those the public key $\vec{h}$ is defined through a computation that is hard-to-invert without knowledge of $\vec{f}$ or $\vec{g}$, so that the public key does not reveal the private keys.

Precisely, $\vec{h} = (h_0, \ldots, h_{N-1})$ is defined by the relation $h(x) = p \cdot f^{-1}(x) * g(x)$ in the ring R_q (for some p co-prime with q and much smaller than it), thus is a vector with entries in C_q for a given $q \in \mathbb{N}_{>0}$. Except for $\vec{f}$ and $\vec{g}$, the other parameters are considered public. In matrix form we may express the relation between the keys by saying $\vec{f} \cdot (p^{-1}\mathbf{H}) = \vec{g} \mod q$, where $\mathbf{H}$ is defined as the circulant matrix containing the coefficients of the polynomial h and its rotations (see [Silverman et al. 2008, Section 6.11.1]).

The reduction modulo q implies the existence of $\vec{u} \in \mathbb{Z}^N$ satisfying $\vec{f} \cdot (p^{-1}\mathbf{H}) - q\vec{u} = \vec{g}$. Fixing $n = 2N$ and given $\vec{h}$ and q, we consider the following as a basis, represented here as a block of four $N \times N$ square matrices, of an NTRU lattice L of dimension n:

$$B = \begin{pmatrix} q\mathbf{I} & \mathbf{0} \\ p^{-1}\mathbf{H} & \mathbf{I} \end{pmatrix},$$

where $\mathbf{I}$ is the identity matrix and $\mathbf{0}$ is the matrix with all entries '0'. And thus we have $(-\vec{u}, \vec{f}) \cdot B = (\vec{g}, \vec{f})$.

Consequently $(\vec{g}, \vec{f})$ is a point of L, of coordinates $(-\vec{u}, \vec{f})$ in basis B, and, considering its entries are all in the set $\{-1, 0, 1\}$, it is likely a shortest non-zero vector of L. Also, it is established in [Hoffstein 2008, Section 7.10.2] that if you pick $\vec{g}$ and $\vec{f}$, rotate their entries, each by the same amount and in the same direction, concatenate them into a new $(\vec{g'}, \vec{f'})$ pair, then this vector is also in L. In addition, by the defining properties of a lattice, so is the negative of each of these rotated vectors. And each of these n non-zero vectors in $\{-1, 0, 1\}^n \cap L^*$ will serve as a decryption key.

This way, discovering $\vec{f}$ or $\vec{g}$ from the public parameters can be reduced to solving the SVP for a particular lattice of dimension n. The search for a short non-zero vector in a lattice $L(B)$ can be understood as finding the set of coordinates $\vec{x} \in \mathbb{Z}^n \setminus \vec{0}$ that minimizes $\|\vec{x}B\|$ (as any lattice vector is an integer linear combination of the basis vectors and vice-versa) and thus SVP is inherently an optimization problem. In this model, when working with any directed search-based algorithm, the natural approach is to make the quality value of a lattice point related to how short it is.

2.3. Genetic Algorithms

Genetic algorithms (GA) are a class of population-based optimization algorithms inspired by the mechanism of evolution, reproduction, and survival of the fittest individuals. GAs perform iterations, evolving a set of approximate solutions — called individuals — until the exact (or a good enough) solution has been found. It is an essential part of genetic algorithms to associate to each individual a fitness value used to quantify the quality of the solutions and to make the current set of solutions interact with one another to derive a new set of solutions. The fitness function must be appropriately defined in that: (i) The individuals within the search space will maximize the fitness if, and only if, they are exact solutions to the underlying problem, and (ii) there is a general trend that the better the approximation to the exact solution the greater the fitness value.

Underlying any GA, there are (typically random) sub-procedures of crossover and mutation. Informally speaking, crossover is about generating a third individual from a pair of previous individuals in a way the new individual bears resemblance to its parents. Mutation is about replacing an individual by another that is different although expected to be similar. Local search procedures were developed independently from GAs, but can be integrated into it. A local search procedure consists of repeatedly replacing a given solution by another, in the neighborhood of the current one (to be conveniently defined in the next section), that maximizes the fitness value until this procedure is no longer able to improve the current solution.

The work of [Luan et al. 2020] deviates a bit from [Ding et al. 2015] for not including a local search step. On the other hand, it does selection through a deterministic process and includes a random restart step to avoid stagnation. To a more detailed description we refer to their articles in the references section.

3. Tailoring a GA to attack NTRU

Our approach follows the idea of using a GA to find SVP solutions but differs from the previously developed algorithms in that our goal is only to solve NTRU lattices. That allow us to operate on the additional assumption that the exact solutions belong to the set $S = \{-1, 0, 1\}^n \setminus \vec{0}$ which we can use as our search space, that is, the set of individuals we consider as approximate or exact solutions. This forces us to modify the definition of fitness as, unlike the others, we do not operate by iteratively finding shorter vectors in L but by minimizing, within S, the distance to L. Thus we shift the paradigm from searching a short vector in L^* to searching in S a vector in L, since the set $S \cap L$ must contain all SVP solutions. This is a core difference in the structure of our GA algorithm from the previous ones and as a consequence our search space is mostly comprised of vectors outside L.

While we could directly apply [Ding et al. 2015] or [Luan et al. 2020]'s algorithm to find short vectors in any NTRU lattice, we want to investigate if its possible to take any advantage on the specificity of the problem and come up with a new algorithm that is more efficient for the subclass of SVP problems in NTRU lattices.

Given a basis B of an NTRU lattice L, if we consider the vector space E spanned by the vectors in B, each point $\vec{v} \in E$ has coordinates $\vec{x} = \vec{v}B^{-1} \in \mathbb{R}^n$, the subset of E with all integer coordinates represent the lattice L. Let the fractional coordinates of any $\vec{v} \in E$ be the vector $\zeta_B(\vec{v}) = \vec{v}B^{-1} - \lfloor \vec{v}B^{-1} \rceil$, where $\lfloor \vec{x} \rceil$ refers to the vector one gets by rounding to the nearest integer every entry of $\vec{x}$. Then the values of $\zeta_B(\vec{v})$ describe what we call a path from $\vec{v}$ to some point of L. More precisely, if $\zeta_B(\vec{v}) = (\zeta_B(\vec{v})_1, \ldots, \zeta_B(\vec{v})_N)$ then starting from $\vec{v}$ if one moves exactly $\zeta_B(\vec{v})_i \cdot \|\vec{b_i}\|$ in the direction of $\vec{b_i}$, for every $i \in \{0, \ldots, N-1\}$, the final destination is the point $\lfloor \vec{v}B^{-1} \rceil B \in L$.

We conclude $\|\zeta_B(\vec{v})\|$ is a proxy measure of the distance from $\vec{v}$ to L and the length of the path they describe is an upper limit for the distance from $\vec{v}$ to L. Note that if the basis vectors are closer to being orthogonal this measure becomes more accurate because, in the degree to which they are parallel, moving in one direction can undo moving in another direction. Due to the fractional coordinates dependency on the choice of basis B this proxy measure can be improved by the beforehand application to B of a basis reduction algorithm such as LLL or BKZ.

We define the functions $cost_B(\vec{v}) = \|\zeta_B(\vec{v})\|^2$ and $fitness_B(\vec{v}) = cost_B(\vec{v})^{-1}$. One may consider $fitness_B(\vec{v}) = \infty$ if $cost_B(\vec{v}) = 0$ but this is not strictly necessary as our algorithm terminates once it finds a zero cost vector.

To form the first generation, we first generate $2n$ vectors drawn from S according to a uniform distribution. The procedures of selection, crossover, mutation and local search are used (in this order) to generate the next generation individuals that will replace the current ones.

When it comes to selecting a pair $(\vec{s}, \vec{t})$ for the crossover operation, we choose two

distinct elements of the population with probability proportional to their fitness.

The crossover operation is defined as follows: Given $\vec{s} = (s_0, \ldots, s_{n-1})$ and $\vec{t} = (t_0, \ldots, t_{n-1})$, both in S, crossover$(\vec{s}, \vec{t}) = \vec{u} = (u_0, \ldots, u_{n-1})$ where each u_i is calculated independently as the realization of a uniform random variable in $\{s_i, t_i\}$. So the amount of genes that $\vec{u}$ inherits from each parent is a binomial variable $Bin(n, 0.5)$.

Let they symbol $\boxplus$ stand for the operation on the set $\{-1, 0, 1\}$: $x \boxplus y = (x+y+1 \mod (3)) - 1$. Once we have the crossover output, the mutation operation is applied to $\vec{u}$ according to the rule: For each u_i, take only one of these actions: with probability $1/n$ make $u_i \leftarrow u_i \boxplus 1$, with probability $1/n$ make $u_i \leftarrow u_i \boxplus 2$, with probability $1 - 2/n$ leave it unchanged. If it happens that, after mutation, the resulting vector is not in S then using the same selected pair, redo the crossover and mutation operations. This ensures the output after mutation is in S.

Now, taking this updated $\vec{u}$ we improve it through a local search on S. We define a neighborhood of $\vec{u}$, $N(\vec{u})$, as the set containing only $\vec{u}$ and the points in S that differ from $\vec{u}$ on a single coordinate, that is, the points within a Hamming distance of 1 from $\vec{u}$. The local search consists of repeatedly updating $\vec{u}$ to one point in $N(\vec{u})$ with the smallest cost until this step can no longer improve the solution.

This process of selection of pair, applying crossover, mutation and then local search are used to generate the majority of individuals that will compose the new population. Keeping one or more of the fittest of one generation to the next is known as an "elitist strategy" and has been used in GAs for many different applications. This strategy guarantees that the maximum fitness observed in the current generation is no lower than that of the previous and has been used, in a different way in both [Ding et al. 2015] and [Luan et al. 2020]. In our case, following the standard of Ding et al., we always keep only the best of each generation.

In this way, we generate population after population until we meet the stopping criteria: A population having an individual with cost zero, that is, a point in L. To the degree that lower cost values correlate with proximity to L, our algorithm tends to replace current solutions for newer solutions that tend to be closer to optimal ones.

3.1. Experimental evaluation of the cost function

Our initial investigation of the cost function examined its behavior near the key $(\vec{g}, \vec{f})$ and observed whether the resulting cost varied only slightly — a characteristic desirable for optimization methods.

We also observed that this local behavior differs depending on whether the lattice basis is BKZ-reduced. Preliminary results show that with a BKZ-reduced basis, the cost function becomes smoother around the optimum. This smoothness may benefit optimization algorithms by providing more consistent gradients or search directions. We have seen this behavior when increasing the NTRU lattice dimension and we report in Figure 1 the results for a lattice of dimension $n = 200$, as $N = 100$ for NTRU. It is worth observing that when the basis is not BKZ-reduced, altering coordinates corresponding to the coefficients of the g polynomial changes slightly the cost function, but altering the coefficients of the f polynomial changes significantly (see the Blue clusters of small and big values in Figure 1(a)).



(a) Histogram (Full Range) **(b) Histogram (Filtered [0, 1])**

Figure 1. Cost values for points in the neighborhood of $(\vec{f}, \vec{g})$, i.e., vectors that differ at only one coordinate. Using the original non-BKZ-reduced basis VS using a BKZ-reduced basis (block size = 10) of dimension $n = 200$.

When expanding our analysis to a range of Hamming distances (Figure 2) we observe that without any basis reduction the correlation between Hamming distance from the secret key and the cost value of a point is weak and the distribution basically stagnates after Hamming distance 5. With LLL or BKZ-reduction, however, if we interpolate the medians of every box-plot we would reach something close to a concave log-like function that slowly increases the median cost values with the Hamming distance. For a lattice of dimension 100, the block size parameter of the BKZ-reduction or whether we use LLL instead did not seem to strongly affect the distribution of cost values by their Hamming distance from the secret key.

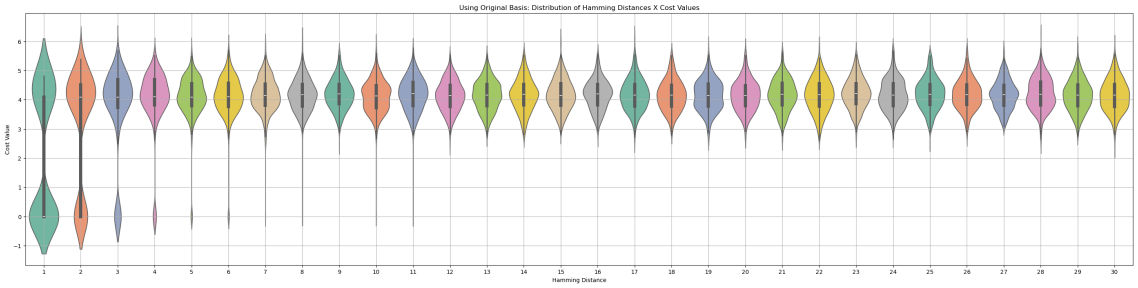
4. Conclusion

The use of genetic algorithms, as well as other metaheuristic methods, have proven to be useful in solving problems of combinatorial nature, including SVPs. The properties of an NTRU lattice allow a redesign of the previously published GAs that solve this problem.

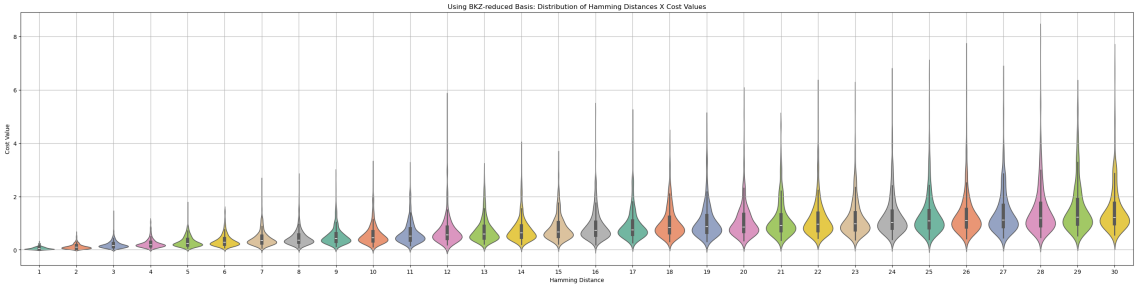
We have shown it is possible to use a search space that is not solely made of lattice points to find the shortest vector in a lattice by constructing an algorithm that operates that way. Although in this article we only focus on NTRU lattices, the core ideas behind our algorithm could in the future extend to solving other types of lattices $L \subset \mathbb{Z}^n$, for some $n \in \mathbb{N}$, by conveniently modifying the definition of the search space, S, to be a subset of $\mathbb{Z}^n$ that contains SVP solutions. That is, as long as we can define the search space S to contain the SVP solutions, and preferably no other lattice vectors and make it as small as possible, we can apply the exact same principles.

NTRU lattices are good candidates to test our ideas as we know that the SVP solutions have entries in $\{-1, 0, 1\}$. So it allows for a straightforward definition of S. Future research questions for this work include: Are there better cost functions to use? Is it practical to reduce the search space to just the coordinates of f, by computing a candidate g using the public key h? Can we beat other GA algorithms when it comes to NTRU lattices SVPs? If we get a competitive GA for solving SVP in NTRU lattices, will a generalization of this algorithm for general lattices remain competitive?

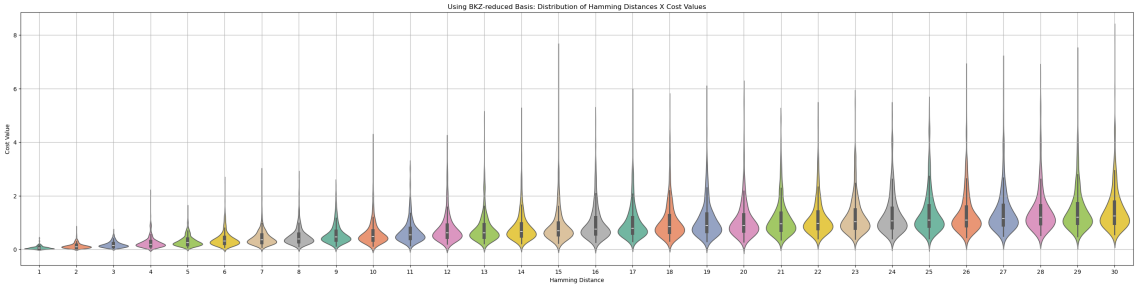
The source code is available at `https://github.com/RaulCA/Genetic-Algorithm-for-SVP-in-NTRU-lattices/`.



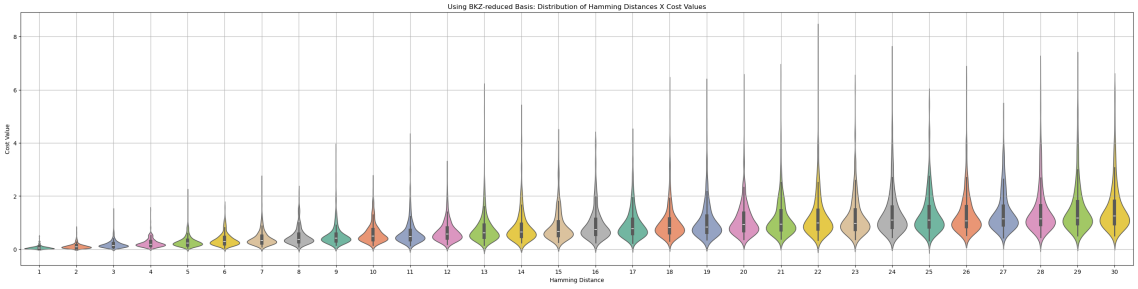
(a) No reduction



(b) With LLL reduction



(c) With BKZ-reduction (block size = 3)



(d) With BKZ-reduction (block size = 50)

Figure 2. Violin plots of how the cost function increases as we move away from the secret pair $(\vec{g}, \vec{f})$ in a lattice of dimension $n = 100$. Applying LLL or BKZ-reduction makes the correlation better between higher costs and greater Hamming distances from the exact solution. For each hamming distance we used 500 samples.

References

Ajitha, S. K., Biswas, S., and Kurur, P. P. (2011). Metropolis algorithm for solving shortest lattice vector problem (svp). In *2011 11th International Conference on Hybrid Intelligent Systems (HIS)*, pages 442–447. IEEE.

Darmstadt lattice challenge. Svp challenge hall of fame. `https://www.latticechallenge.org/svp-challenge/halloffame.php`. Accessed: 2025-04-22.

Ding, D. and Zhu, G. (2014). A random sampling algorithm for svp challenge based on y-sparse representations of short lattice vectors. In *2014 Tenth International Conference on Computational Intelligence and Security*, pages 425–429. IEEE.

Ding, D., Zhu, G., and Wang, X. (2015). A genetic algorithm for searching the shortest lattice vector of svp challenge. In *Proceedings of the 2015 annual conference on genetic and evolutionary computation*, pages 823–830.

Fukase, M. and Kaminaga, M. (2022). Improving genetic algorithms for solving the svp: focusing on low memory consumption and high reproducibility. *Iran Journal of Computer Science*, 5(4):359–372.

Fukase, M. and Kashiwabara, K. (2015). An accelerated algorithm for solving svp based on statistical analysis. *Journal of Information Processing*, 23(1):67–80.

Hoffstein, J. (2008). An introduction to cryptography. In *An Introduction to Mathematical Cryptography*, pages 1–58. Springer.

Hoffstein, J., Pipher, J., and Silverman, J. H. (1998). Ntru: A ring-based public key cryptosystem. In *International algorithmic number theory symposium*, pages 267–288. Springer.

Luan, L., Gu, C., and Zheng, Y. (2020). A genetic algorithm with restart strategy for solving approximate shortest vector problem. In *2020 12th International Conference on Advanced Computational Intelligence (ICACI)*, pages 243–250. IEEE.

Moghissi, G. R. and Payandeh, A. (2019). A parallel evolutionary search for shortest vector problem. *International Journal of Information Technology and Computer Science*.

Reddy, V. D. and Rao, G. (2021). Meta-heuristic approaches to solve shortest lattice vector problem. *Journal of Discrete Mathematical Sciences and Cryptography*, 24(1):81–91.

Silverman, J. H., Pipher, J., and Hoffstein, J. (2008). *An introduction to mathematical cryptography*, volume 1. Springer.

Avaliação de OCR Embarcado para Dispositivos Móveis: Desempenho, Privacidade e Usabilidade

João Carlos Nepomuceno Fernandes*, Francisco Erick Souza Gomes*, Douglas de Araújo Rodrigues†,
Vinícius Lagrota Rodrigues da Costa‡, Paulo Antônio Leal Rego†,
José Antonio Fernandes de Macedo† and Pedro Pedrosa Rebouças Filho*
*Instituto Federal de Educação, Ciência e Tecnologia do Ceará
†Universidade Federal do Ceará
‡Centro de Pesquisa e Desenvolvimento para a Segurança das Comunicações (CEPESC)
Email: {joaocarlos, ericksouza, douglas, jose.macedo}@insightlab.ufc.br,
vinicius.1232@dte.gov.br, pedrosarf@ifce.edu.br

***Abstract*—This study presents a comparative analysis between three OCR technologies for mobile devices: Vision OCR, MLKit Mobile, and Tesseract Mobile. A total of 848 real examples of words extracted from urban environments were used, and the tools were evaluated with and without the application of preprocessing techniques. The metrics used enabled a complete evaluation of the performance of each solution. The results show that Vision OCR performs best, followed by MLKit and Tesseract with optimized preprocessing. Security aspects (processing on the device), usability (ease of operation, installation, and documentation, considering file size in embedded environments), and ease of integration were also analyzed. Native solutions offer superior performance with less need for adjustments, while Tesseract, despite requiring a more sophisticated pipeline, ensures maximum privacy because it is open source and controlled by the developer.**

***Resumo*—Este estudo apresenta uma análise comparativa entre três tecnologias de OCR para dispositivos móveis: *Vision OCR, MLKit Mobile e Tesseract Mobile*. Foram utilizados 848 exemplos reais de palavras extraídas de ambientes urbanos, e as ferramentas foram avaliadas com e sem a aplicação de técnicas de pré-processamento. As métricas utilizadas possibilitaram uma avaliação completa do desempenho de cada solução. Os resultados mostram que o *Vision OCR* apresenta o melhor desempenho, seguido pelo *MLKit* e pelo *Tesseract* com pré-processamento otimizado. Também foram analisados aspectos de segurança (processamento no dispositivo), usabilidade (facilidade de operação, instalação e documentação, considerando o tamanho dos arquivos em ambientes embarcados) e facilidade de integração. As soluções nativas oferecem desempenho superior com menor necessidade de ajustes, enquanto o *Tesseract*, apesar de exigir um pipeline mais sofisticado, garante privacidade máxima por ser de código aberto e controlado pelo desenvolvedor.**

I. Introdução

A popularização dos dispositivos móveis transformou a forma como interagimos com o mundo, conectando o ambiente físico ao digital. Nesse contexto, o Reconhecimento Óptico de Caracteres (*Optical Character Recognition*, OCR) consolidou-se como ferramenta essencial, aplicada desde a automação de dados até a tradução instantânea e o suporte a pessoas com deficiência visual [1]–[3].

Apesar de sua relevância, implementar OCR em dispositivos móveis apresenta desafios, sobretudo quando se busca execução totalmente *on-device*, evitando a exposição de dados a serviços externos. Soluções como o *Vision*, da Apple [4], e o *ML Kit*, do Google [5], integram modelos de *deep learning* para detecção e reconhecimento de texto. Já o *Tesseract Mobile* [6], [7], embora eficaz no reconhecimento, depende de pipelines adicionais para detecção, exigindo pré-processamento ou integração com outros modelos.

A literatura frequentemente compara motores de OCR em ambientes de desktop ou via APIs, mas poucos estudos exploram soluções móveis com processamento local [8], [9]. São ainda mais raras as avaliações comparativas sob condições uniformes em dispositivos reais. Trabalhos como o de Silva *et al.* [10] e Correa *et al.* [11] propõem estratégias complementares, mas não contemplam execução inteiramente embarcada.

As tecnologias analisadas neste estudo — *Vision*, *ML Kit* e *Tesseract* — foram escolhidas pela viabilidade em aplicações móveis: o primeiro otimizado para iOS, o segundo integrado ao Android e o terceiro, por ser de código aberto, permitindo maior controle e privacidade. Modelos recentes baseados em *PyTorch* ou *TensorFlow Lite* não foram considerados devido ao custo de deployment, maior complexidade de integração e aumento no tamanho do aplicativo, demandando otimizações adicionais [12].

Neste trabalho, conduzimos uma análise comparativa de três soluções de OCR *on-device*, avaliando sua precisão em um dataset de imagens reais com regiões de texto previamente delimitadas. Também investigamos o impacto de técnicas de pré-processamento local, com todas as etapas executadas no próprio dispositivo. Além dos resultados quantitativos, discutimos aspectos práticos de integração, ressaltando a preservação da privacidade como benefício da execução local. Esse aspecto é particularmente relevante em aplicações críticas, como o processamento de prontuários médicos, comprovantes bancários ou documentos oficiais, onde a confidencialidade dos dados deve ser preservada.

II. Trabalhos relacionados

A literatura apresenta diversas análises comparativas de tecnologias de OCR. Estudos como os de Malkadi *et al.* [13] e Tafti *et al.* [14] demonstram consistentemente que soluções comerciais e baseadas em nuvem, como as oferecidas pelo

Google, tendem a superar o *Tesseract* em acurácia para tarefas genéricas de extração de texto. Essa tendência é reforçada em estudos de caso específicos, como o de Thammarak *et al.* [15], que ao analisar certificados de veículos em tailandês, obteve 84.43% de acurácia com a *API Google Cloud Vision* contra 47.02% do *Tesseract*. Da mesma forma, Kaur e Sharma [16] observaram um desempenho superior da *API do Google* para o reconhecimento de texto em Punjabi.

Apesar do desempenho inferior em comparações diretas, a eficácia do *Tesseract* pode ser melhorada através da adição de um *pipeline* de pré-processamento nas imagens. O impacto dessa etapa é um tema central na pesquisa sobre OCR, com diversos estudos demonstrando que a aplicação dessas técnicas pode otimizar o reconhecimento em diferentes contextos, desde textos manuscritos até documentos em idiomas específicos [7], [17], [18].

Além disso, trabalhos como o de Sporici *et al.* [19] mostram que técnicas, como o uso de filtros convolucionais otimizados por aprendizagem por reforço, podem aumentar a acurácia do *Tesseract* 4.0 em mais de 300% em datasets desafiadores. Outros estudos, como o de Michalak e Okarma [20], exploram métodos específicos como a filtragem por entropia local para aprimorar a binarização de imagens com iluminação irregular, uma condição particularmente comum e problemática em capturas realizadas por dispositivos móveis.

No entanto, a literatura carece de estudos que comparem diretamente as soluções nativas dos sistemas operacionais móveis dominantes com o *Tesseract* em um cenário de uso realista e com processamento *on-device*. O *Vision*, *framework* oficial da Apple — isto é, um conjunto de ferramentas e bibliotecas de alto nível disponibilizado pelo sistema operacional para tarefas específicas de visão computacional — é amplamente utilizado por aplicativos iOS para OCR, realizando o processamento localmente no próprio dispositivo [4].

O *ML Kit*, disponibilizado pelo Google, é compatível tanto com Android quanto com iOS, mas apenas no Android oferece suporte a reconhecimento de texto *on-device*; no iOS, essa funcionalidade depende de processamento em nuvem [5]. Ambos os *frameworks* incluem, além do reconhecimento de caracteres, mecanismos de detecção automática de regiões de texto na imagem — uma etapa que o *Tesseract* não executa de forma autônoma, exigindo segmentação prévia ou o uso de pipelines externos.

Neste trabalho, no entanto, o foco está na etapa de classificação dos caracteres reconhecidos (*text recognition*), e não na detecção das regiões textuais, conforme refletido no dataset utilizado. Muitas comparações existentes se concentram em versões de desktop ou APIs comerciais em nuvem, negligenciando as particularidades do ambiente *mobile* e os desafios específicos de desenvolvimento.

Além disso, poucos trabalhos combinam a análise de acurácia com fatores práticos cruciais para desenvolvedores, como a facilidade de integração e as implicações de segurança e privacidade associadas ao processamento local. Este artigo visa preencher essa lacuna, oferecendo uma análise multifacetada que não apenas mede o desempenho bruto, mas também avalia a adequação de cada solução ao ecossistema de desenvolvimento móvel.

Recentemente, surgiram modelos leves de OCR baseados em *deep learning*, implementados em *frameworks* como *TensorFlow Lite* e em arquiteturas como *MobileNet*. Os trabalhos de Franco *et al.* [21] e Sari *et al.* [22] abordam a utilização desses modelos para o referente contexto. Apesar de demonstrarem bom desempenho em *benchmarks* acadêmicos, sua adoção em cenários de produção ainda é limitada por fatores práticos: falta de integração nativa em iOS e Android, necessidade de manutenção manual dos pesos de rede e impacto considerável no tamanho do aplicativo final. Diferentemente disso, *frameworks* como *Vision* e *ML Kit* oferecem APIs já integradas ao sistema operacional, simplificando o desenvolvimento e garantindo suporte contínuo.

III. Metodologia

Esta seção descreve detalhadamente o conjunto de dados utilizado, os métodos empregados para o reconhecimento de texto e as métricas adotadas para avaliação dos resultados. A Figura 1 ilustra o fluxo do processo experimental, incluindo as etapas de pré-processamento e avaliação dos modelos de OCR. A Figura 2, por sua vez, apresenta exemplos de amostras contidas no conjunto de dados utilizado.

A. Dataset

O *dataset* utilizado é uma versão adaptada do *ICDAR 2003 Robust Reading Dataset* [23], composta por 848 imagens de palavras isoladas com caracteres do alfabeto latino e algarismos decimais, coletadas em ambientes urbanos reais. Diferente do original, que contém cenas completas, esta versão foca em regiões de texto já segmentadas, simulando a saída de um detector de textos e permitindo avaliar especificamente o reconhecimento de caracteres.

Datasets mais recentes não foram utilizados por apresentarem textos em múltiplos idiomas, exigindo uma pré-seleção manual, e porque muitas imagens não estavam recortadas, o que demandaria ferramentas de *cropping* da ROI, tarefa fora do escopo deste estudo. O *ICDAR 2003*, embora antigo, já foi amplamente validado em trabalhos de OCR, mas não em cenários embarcados, justificando sua escolha para avaliação prática *on-device*. Esta escolha implica uma limitação conhecida, que será abordada em trabalhos futuros com a inclusão de datasets contemporâneos e mais complexos.

B. Metodologia de avaliação

Este estudo adotou uma abordagem sistemática para avaliar o desempenho das soluções utilizadas, dividida em duas fases:

1) Pré-processamento: Foram implementadas cinco técnicas de pré-processamento em imagens, cada uma com características específicas:

- Original: Imagem base sem modificações, servindo como referência para comparação dos demais métodos.
- Conversão para Escala de Cinza com equalização de histograma: Transformação não-linear do espaço de cores RGB para tons de cinza utilizando coeficientes

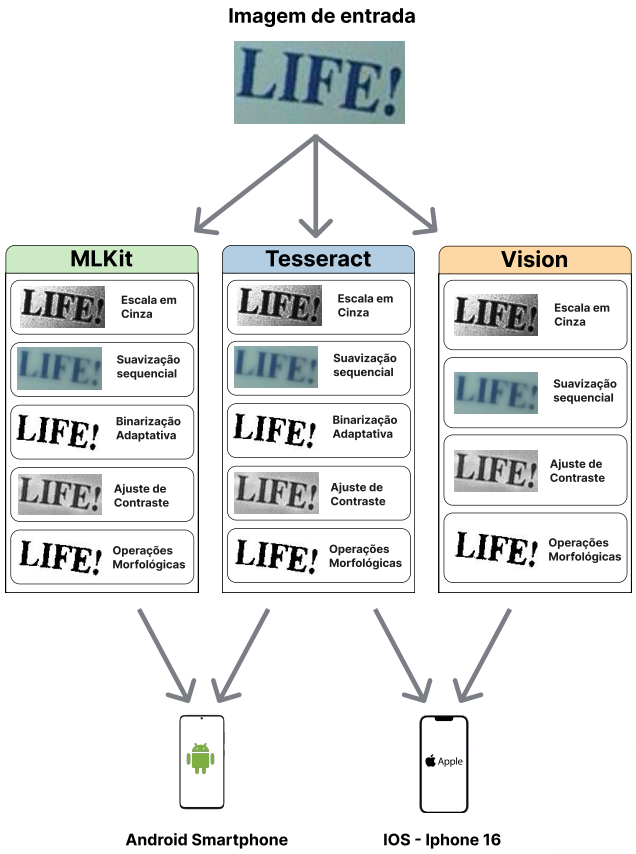


Figura 1. Fluxograma da metodologia adotada para avaliação dos motores de OCR (*MLKit, Tesseract e Vision*) em dispositivos Android e iOS. A imagem de entrada é submetida a diferentes técnicas de pré-processamento. Cada técnica é aplicada de forma independente antes da execução do OCR, permitindo verificar o impacto individual de cada etapa no desempenho de reconhecimento em cada motor e sistema operacional.



Figura 2. Exemplos de amostras contidas no dataset utilizado.

de luminância otimizados (Y = 0.299R + 0.587G + 0.114B), seguida de equalização de histograma para melhor distribuição tonal.

- Suavização sequencial: Aplicação sequencial de filtro Gaussiano (kernel 3×3, σ=0.5) para suavização de alta frequência, seguido por filtro bilateral (d=9, σ_{color}=75, σ_{space}=75) para preservação de bordas textuais.
- Ajuste de Contraste : Equalização adaptativa de histograma (*CLAHE*), com tamanho de *tile* de 8×8 pixels e limite de contraste 2.0, especialmente eficaz para imagens com iluminação irregular (Zimmerman *et al.* [24]).
- Binarização Adaptativa: *Thresholding* automático pelo método de Otsu, que determina iterativamente o limiar ótimo para separação *foreground/background* baseado na distribuição de intensidades dos pixels.
- Operações Morfológicas: Abertura morfológica (erosão seguida de dilatação) com elemento estruturante circular de raio 2 pixels, visando remover artefatos pequenos enquanto mantém a estrutura geral dos caracteres.

A implementação considerou as restrições de cada plataforma: no Android, utilizamos *OpenCV Mobile* combinado com *ML Kit* [5], permitindo a aplicação de todas as técnicas. No iOS, entretanto, o framework *Vision* [4] não oferece suporte nativo à binarização adaptativa, restringindo sua aplicação nesta plataforma e limitando os experimentos a outros métodos de pré-processamento disponíveis. Cada técnica foi aplicada isoladamente para permitir uma análise individual. É importante notar que o *Tesseract* [6] foi aplicado em ambas as plataformas.

2) Inferência dos Modelos OCR: Foram avaliadas três soluções distintas de OCR, com abordagens e arquiteturas variadas:

- *Vision OCR* [4]: API nativa da Apple com processamento local, utilizando o *framework Vision* para reconhecimento de texto em imagens. A inferência ocorre em tempo real, e os resultados são retornados como blocos, linhas e palavras, exigindo *parsing* para extração do texto contínuo.
- *MLKit Mobile* [5]: Biblioteca do Google para dispositivos Android, operando no modo *on-device*(executado localmente). O texto é retornado segmentado por blocos e linhas, de forma similar ao *Vision*, com suporte à múltiplos idiomas. Vale destacar que este *framework* também pode ser utilizado em plataformas iOS; entretanto, nessa configuração, o processamento ocorre via API (remota), e não localmente.
- *Tesseract* [6]: Modelo *open-source* integrado via *wrappers* personalizados para Android e iOS. Foram testadas as configurações de tratamento de uma única linha de texto e *layout* baseado em blocos, esta última foi selecionada por apresentar melhor resultado. A inferência foi realizada em lote, com o OCR aplicado sobre cada imagem pré-processada individualmente.

Todas as soluções foram integradas nativamente aos respectivos aplicativos móveis e configuradas para operar exclusivamente no modo *offline*. As imagens processadas por cada técnica de pré-processamento foram submetidas individualmente a cada modelo de OCR, garantindo que os resultados fossem comparáveis e que os efeitos de cada etapa pudessem ser isolados. Os textos retornados por cada inferência foram armazenados em arquivos estruturados para posterior comparação com os padrões de referência, permitindo o cálculo das métricas de desempenho.

C. Métricas de Avaliação

A avaliação do desempenho das soluções de OCR foi conduzida por meio de quatro métricas comumente utilizadas na literatura: Acurácia por Caractere (*Character Accuracy - CharAcc*), Acurácia por Palavra (*Word Accuracy - WordAcc*), Taxa de Erro por Caractere (*Character Error Rate - CER*) e Taxa de Erro por Palavra (*Word Error Rate - WER*). Apesar das abreviações serem comuns em trabalhos de reconhecimento óptico de caracteres, elas são aqui explicitadas para maior clareza.

A *CharAcc* mede a proporção de caracteres corretamente reconhecidos em relação ao total de caracteres do texto de referência, considerando correspondência exata com letras maiúsculas/minúsculas e símbolos especiais. Essa métrica é definida conforme a Equação 1:

$$CharAcc = \left(1 - \frac{\sum \text{Erros de Caractere}}{\sum \text{Total de Caracteres}}\right) \times 100\% \quad (1)$$

A *WordAcc* quantifica o percentual de palavras que foram reconhecidas com 100% de precisão, sendo uma palavra considerada incorreta mesmo com um único caractere errado. A fórmula é apresentada na Equação 2:

$$WordAcc = \frac{\text{Número de Palavras Perfeitas}}{\text{Total de Palavras}} \times 100\% \quad (2)$$

A *CER* representa a distância de edição entre o texto reconhecido e o texto de referência, somando substituições (S), inserções (I) e deleções (D) de caracteres em relação ao número total de caracteres (N), conforme a Equação 3:

$$CER = \frac{S + I + D}{N} \times 100\% \quad (3)$$

De forma análoga, a *WER* avalia as alterações no nível de palavras completas, considerando substituições (S_w), inserções (I_w) e deleções (D_w) de palavras em relação ao total de palavras (N_w), como mostrado na Equação 4:

$$WER = \frac{S_w + I_w + D_w}{N_w} \times 100\% \quad (4)$$

Essas métricas oferecem perspectivas complementares. Enquanto *CharAcc* e *WordAcc* destacam a taxa de acerto, *CER* e *WER* quantificam a magnitude dos erros. Em cenários práticos de OCR, a *CER* tende a ser mais informativa, pois considera todos os erros individualmente. Já a *WordAcc* é mais rigorosa ao penalizar palavras parcialmente incorretas da mesma forma que palavras completamente erradas.

D. Configuração dos Dispositivos

Os experimentos foram conduzidos em dispositivos móveis sob condições reais de aplicações embarcadas, com processamento integralmente *on-device*.

- Android: Dispositivo virtual *Pixel 7* no *Android Studio*, executando Android 16 (API 36). Configuração equivalente a hardware físico intermediário-avançado com chip *Google Tensor G2*, tela de 6.31"e resolução 1080×2400 px.
- **iOS**: Dispositivo físico *iPhone 16* com *iOS 18* e chip *Apple A18 Pro*, tela de 6.3"e resolução 1179×2556 px.

IV. Resultados e Discussões

Nesta seção, os resultados são apresentados e discutidos de forma integrada, considerando a performance dos modelos de OCR em dispositivos Android e iOS, com foco no impacto das técnicas de pré-processamento. A análise é baseada nas métricas *WordAcc*, *CharAcc*, *WER* e *CER*, previamente definidas na Seção de Metodologia.

A. Resultados por Plataforma e Tecnologia

A Tabela I apresenta os resultados obtidos na plataforma Android utilizando os modelos *MLKit* e *Tesseract* com diferentes pré-processamentos aplicados.

Tabela I
Resultados no Android: *MLKit* e *Tesseract*

Modelo	Processamento	WordAcc (%) ↑	CharAcc (%) ↑	WER (%) ↓	CER (%) ↓
MLKit	Tons de Cinza	80.19	89.48	19.81	10.52
	Original	80.07	89.82	19.85	10.18
	Suavização sequencial	76.30	87.78	23.70	12.22
	Ajuste de Contraste	76.06	86.06	23.94	13.94
	Limiarização Adaptativa	54.72	73.02	45.28	26.98
	Morfologia	48.70	69.86	51.30	30.14
Tesseract	Tons de Cinza	63.68	77.33	36.32	22.67
	Original	59.79	74.70	40.21	25.30
	Suavização sequencial	58.45	72.83	41.55	27.17
	Ajuste de Contraste	58.25	72.26	41.75	27.74
	Morfologia	42.37	62.89	57.63	37.11
	Limiarização Adaptativa	31.37	50.96	68.63	49.04

Legenda: ■ Melhor resultado de cada tecnologia. Setas indicam se valores maiores (↑) ou menores (↓) são desejáveis.

No Android, observa-se que o *MLKit* se destaca na configuração original, sendo ligeiramente superado pela versão em tons de cinza. Isso sugere que o modelo se beneficia de pré-processamentos leves, mas é sensível a alterações mais invasivas, como morfologia e binarização adaptativa, que causaram perdas de desempenho. O *Tesseract*, por sua vez, parte de um desempenho inferior na imagem original, mas apresenta melhora notável com a aplicação de pré-processamento em escala de cinza, o que evidencia o potencial de otimização mesmo em soluções *open-source* mais simples.Em seguida, a Tabela II apresenta os resultados da plataforma iOS, comparando o modelo nativo *Vision OCR* com o *Tesseract*.

No iOS, o *Vision OCR* apresentou o melhor desempenho. Pré-processamentos adicionais não trouxeram melhorias, e em alguns casos, como ajuste de contraste, que reduziu a acurácia. O *Tesseract* teve desempenho inferior, no original, e reduções

Tabela II
RESULTADOS NO IOS: VISION E TESSERACT

Modelo	Processamento	WordAcc (%) ↑	CharAcc (%) ↑	WER (%) ↓	CER (%) ↓
Vision OCR	Original	86.79	91.38	13.21	08.62
	Tons de Cinza	81.13	89.53	20.17	09.15
	Suavização sequencial	80.90	87.11	19.04	12.89
	Ajuste de Contraste	61.67	68.95	38.27	31.05
	Morfologia	48.26	57.72	51.74	42.28
Tesseract	Original	56.81	65.47	43.19	34.53
	Suavização sequencial	44.10	53.57	55.90	46.43
	Ajuste de Contraste	44.78	51.29	55.22	48.71
	Tons de Cinza	34.11	56.00	73.64	38.64
	Morfologia	32.41	36.75	69.48	60.89

Legenda: ■ Melhor resultado de cada tecnologia. Setas indicam se valores maiores (↑) ou menores (↓) são desejáveis.

significativas na maioria dos pré-processamentos, indicando baixa adaptação à plataforma.

De forma geral, o *Vision OCR* se confirma como a solução mais eficaz no iOS, enquanto o *MLKit* mantém boa performance no Android, especialmente com ajustes simples como tons de cinza. O *Tesseract*, embora mais limitado, obtém ganhos pontuais com pré-processamentos leves, podendo ser viável em cenários locais e *offline*.

B. Impacto dos Modelos Embarcados no Aplicativo

O tamanho dos modelos utilizados é um ponto relevante em aplicações móveis, especialmente porque, neste estudo, eles são embarcados no próprio aplicativo. Essa abordagem garante que a funcionalidade opere sem conexão com a internet, mas exige atenção devido ao impacto no tamanho final do app e na usabilidade — entendida como a facilidade de instalação, operação e manutenção da solução, sobretudo em ambientes totalmente embarcados.

No iOS, o *Vision*, solução nativa da Apple para OCR, utiliza modelos de *machine learning* já integrados ao sistema operacional. Ao usar funcionalidades como `VNDocumentCameraViewController` ou `VNRecognizeTextRequest`, o aplicativo acessa esses modelos diretamente, sem aumentar o tamanho do binário, garantindo funcionalidade *offline* imediata, sem *downloads* adicionais, e mantendo o app leve e simples de instalar e usar.

Já no Android, o *ML Kit* adota uma estratégia diferente. Embora também permita uso *offline*, o desenvolvedor pode optar por embarcar os modelos diretamente no aplicativo, o que aumenta o tamanho do APK ou AAB inicial, mas elimina a latência de *download* e assegura a disponibilidade imediata da funcionalidade após a instalação, adicionando cerca de 300 *kilobytes* ao pacote [5]. Por fim, o *Tesseract* requer uma inclusão mais explícita dos componentes, como o motor principal e os arquivos de idioma, que podem adicionar de 5 a 15 *megabytes* ou mais por idioma, além do tamanho do motor compilado [25]. Essa abordagem oferece controle total ao desenvolvedor, mas demanda gerenciamento manual dos componentes e equilíbrio entre tamanho e funcionalidade, o que pode afetar a usabilidade.

Em síntese, o *Vision Framework* proporciona uma solução nativa sem impacto no tamanho do app e com usabilidade favorecida, o *ML Kit* oferece embarque opcional de modelos com impacto moderado e instalação simplificada, enquanto o *Tesseract* garante maior controle, porém com maior esforço de gestão para preservar a experiência do usuário.

C. Segurança e Privacidade no Processamento On-Device

Do ponto de vista da segurança dos dados, todas as soluções avaliadas oferecem suporte ao processamento *on-device*, ou seja, sem envio das imagens para servidores externos — característica essencial para aplicações sensíveis, como leitura de documentos ou dados pessoais.

Entretanto, o nível de controle sobre a privacidade varia. O *Tesseract Mobile* é o mais seguro nesse quesito: por ser uma biblioteca *open-source* e executada integralmente no dispositivo, nenhum dado sai do ambiente local, e não há dependência de políticas de terceiros. O *Vision OCR OCR* também opera de forma totalmente local, dentro do ecossistema Apple, com fortes garantias de privacidade, mas com menor flexibilidade sobre o pipeline interno. Já o *MLKit Mobile*, embora também suporte modo *on-device*, é parte do ecossistema Google, o que requer atenção redobrada às configurações de nuvem e políticas de privacidade para evitar o envio involuntário de dados.

Embora existam propostas como a de Correa *et al.* [11], que visam aprimorar o reconhecimento em documentos complexos, essas abordagens normalmente dependem de processamento externo ou combinação de serviços na nuvem. O presente estudo, ao se concentrar na execução local e autônoma dos modelos, elimina esses riscos e reforça a importância da computação de borda para a preservação da privacidade.

V. CONCLUSÃO

Este trabalho avaliou comparativamente o desempenho de diferentes modelos de OCR em dispositivos móveis de plataformas Android e iOS, considerando o impacto de diversas técnicas de pré-processamento de imagem. Os resultados demonstraram que soluções nativas como o *Vision OCR OCR e o MLKit Mobile* apresentam desempenho superior em suas configurações originais ou com ajustes mínimos na entrada.

Destaca-se que o *Vision OCR* obteve o melhor resultado geral sem necessidade de pré-processamento, evidenciando a qualidade do *framework* da Apple. O *MLKit* também mostrou-se eficaz, com ganhos discretos por meio de pré-processamento. Por outro lado, o *Tesseract Mobile*, apesar de sua performance inferior, demonstrou evolução com técnicas simples de aprimoramento, o que reforça seu valor em cenários com restrições de privacidade e custo, devido à sua natureza open-source e totalmente local.

Do ponto de vista da segurança dos dados, todas as tecnologias avaliadas oferecem, em geral, suporte ao processamento *on-device*, importante para aplicações que lidam com

informações sensíveis, como documentos e dados pessoais. Contudo, é importante destacar uma exceção: na plataforma iOS, o *MLKit* não realiza OCR localmente, encaminhando as imagens para os servidores do Google para processamento. Essa particularidade impõe limitação na questão de segurança, em cenários que demandam total controle e sigilo dos dados.

Nesse contexto, o *Tesseract* se destaca como a solução mais segura e auditável, por ser completamente *open-source* e executado de forma inteiramente local. Já o Vision OCR, apesar de também operar *on-device*, possui um *pipeline* fechado e menos flexível, ainda que com fortes garantias de privacidade dentro do ecossistema Apple.

A. Trabalhos Futuros

Como próximos passos, propõe-se a exploração de modelos de OCR multilíngue, ampliando a aplicabilidade da solução em cenários reais que envolvem textos em diferentes idiomas, em datasets mais recentes e desafiadores. Outra linha promissora é a avaliação de modelos neurais leves, como implementações em *TensorFlow Lite* e *PyTorch Mobile* presentes no estado da arte, que podem oferecer melhor compromisso entre acurácia e eficiência em dispositivos móveis. Estratégias de otimização de tempo de execução, incluindo técnicas de quantização e *pruning*, também serão consideradas para reduzir o consumo de recursos sem perda significativa de desempenho.

Além disso, vislumbra-se a investigação de modelos de OCR baseados em LLMs, aproveitando a popularização de arquiteturas multimodais para explorar abordagens com *transformers* ou modelos de linguagem visual. Outra direção relevante é a integração com sistemas de validação semântica, especialmente em aplicações críticas como a leitura de documentos oficiais, onde a verificação do conteúdo pode aumentar a confiabilidade dos resultados.

REFERÊNCIAS

[1] B. Chatpaitoon and R. Itthipanichpong, "Effect of optical text recognition (ocr) and text-to-speech (tts) smartphone app on vision-related quality of life in visually impaired people." *Investigative Ophthalmology & Visual Science*, vol. 64, no. 8, pp. 2808–2808, 2023.

[2] S. Kompalli, S. Nayak, S. Setlur, and V. Govindaraju, "Challenges in ocr of devanagari documents," in *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*. IEEE, 2005, pp. 327–331.

[3] M. Laine and O. S. Nevalainen, "A standalone ocr system for mobile cameraphones," in *2006 IEEE 17th international symposium on personal, indoor and mobile radio communications*. IEEE, 2006, pp. 1–5.

[4] Apple Inc., "Vision Framework - Apple Developer Documentation," 2024, acessado em: 30 de julho de 2025. [Online]. Available: https://developer.apple.com/documentation/vision?language=objc

[5] Google Developers, "ML Kit — Google for Developers," 2024, acessado em: 30 de julho de 2025. [Online]. Available: https://developers.google.com/ml-kit?hl=pt-br

[6] R. Smith, "An overview of the tesseract ocr engine," in *Ninth international conference on document analysis and recognition (ICDAR 2007)*, vol. 2. IEEE, 2007, pp. 629–633.

[7] S. Badla, "Improving the efficiency of tesseract ocr engine," 2014.

[8] Y. Wang, Y. Liu, T. Wu, and I. Duncan, "A cost-effective ocr implementation to prevent phishing on mobile platforms," in *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*. IEEE, 2020, pp. 1–8.

[9] T. Mantoro, A. M. Sobri, and W. Usino, "Optical character recognition (ocr) performance in server-based mobile environment," in *2013 International Conference on Advanced Computer Science Applications and Technologies*. IEEE, 2013, pp. 423–428.

[10] L. V. da Silva, P. L. J. D. Junior, and S. S. da Costa Botelho, "An optical character recognition post-processing method for technical documents," in *Conference on Graphics, Patterns and Images (SIBGRAPI)*. SBC, 2023, pp. 126–131.

[11] I. L. Correa, P. L. J. Drews, and R. N. Rodrigues, "Combination of optical character recognition engines for documents containing sparse text and alphanumeric codes," in *2021 34th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*. IEEE, 2021, pp. 299–306.

[12] Mindee, "doctr: Document text recognition," https://github.com/mindee/doctr, 2021.

[13] A. Malkadi, M. Alahmadi, and S. Haiduc, "A study on the accuracy of ocr engines for source code transcription from programming screencasts," in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 65–75.

[14] A. P. Tafti, A. Baghaie, M. Assefi, H. R. Arabnia, Z. Yu, and P. Peissig, "Ocr as a service: an experimental evaluation of google docs ocr, tesseract, abbyy finereader, and transym," in *International Symposium on Visual Computing*. Springer, 2016, pp. 735–746.

[15] K. Thammarak, P. Kongkla, Y. Sirisathitkul, and S. Intakosum, "Comparative analysis of tesseract and google cloud vision for thai vehicle registration certificate," *International Journal of Electrical and Computer Engineering*, vol. 12, no. 2, pp. 1849–1858, 2022.

[16] R. Kaur and D. V. Sharma, "Punjabi text recognition system for portable devices: A comparative performance analysis of cloud vision api with tesseract," *Journal of Computer Science and Engineering (JCSE)*, vol. 2, no. 2, pp. 104–111, 2021.

[17] L. R. Mursari and A. Wibowo, "The effectiveness of image preprocessing on digital handwritten scripts recognition with the implementation of ocr tesseract," *Computer Engineering and Applications Journal*, vol. 10, no. 3, pp. 177–186, 2021.

[18] U. Hengaju and B. K. Bal, "Improving the recognition accuracy of tesseract-ocr engine on nepali text images via preprocessing," *Advancement in Image Processing and Pattern Recognition*, vol. 3, no. 2, p. 3, 2023.

[19] D. Sporici, E. Cușnir, and C.-A. Boiangiu, "Improving the accuracy of tesseract 4.0 ocr engine using convolution-based preprocessing," *Symmetry*, vol. 12, no. 5, p. 715, 2020.

[20] H. Michalak and K. Okarma, "Improvement of image binarization methods using image preprocessing with local entropy filtering for alphanumerical character recognition purposes," *entropy*, vol. 21, no. 6, p. 562, 2019.

[21] A. Franco, "Optical recognition of the philippines'ancient text: A deep learning approach," *INTERNATIONAL JOURNAL*, vol. 8, no. 03, 2025.

[22] B. P. Sari, R. W. Sholikah, and R. V. H. Ginardi, "The development of mobile application for object recognition based on deep learning to assist people with visually impaired," in *2024 2nd International Symposium on Information Technology and Digital Innovation (ISITDI)*. IEEE, 2024, pp. 222–227.

[23] S. Lucas, A. Panaretos, L. Sosa, A. Tang, S. Wong, and R. Young, "Icdar 2003 robust reading competitions," in *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, 2003, pp. 682–687.

[24] J. B. Zimmerman, S. M. Pizer, E. V. Staab, J. R. Perry, W. McCartney, and B. C. Brenton, "An evaluation of the effectiveness of adaptive histogram equalization for contrast enhancement," *IEEE Transactions on Medical Imaging*, vol. 7, no. 4, pp. 304–312, 1988.

[25] Tesseract OCR, "Data Files — Tesseract OCR," https://tesseract-ocr.github.io/tessdoc/tess3/Data-Files.html, 2025, acessado em 8 de agosto de 2025.

Performance Evaluation of Cloud Native Applications: A Systematic Mapping Study

Francisco Airton Silva[1], Fernando A. M. Trinta[2], Michel S. Bonfim[2], José Antonio F. de Macedo[2], Paulo A. L. Rego[2], Vinícius Lagrota[3]

[1] Federal University of Piauí (UFPI), Picos, Piauí, Brazil.
[2] Federal University of Ceará (UFC), Fortaleza, Ceará, Brazil.
[3] Research and Development Center for Communication Security (CEPESC), Brasília, Distrito Federal, Brazil.

Contributing authors: faps@ufpi.edu.br; fernando.trinta@dc.ufc.br; michelsb@ufc.br; jose.macedo@dc.ufc.br; paulo@dc.ufc.br; vinicius.1232@abin.gov.br;

Abstract

Cloud-native applications leverages cloud environments for enhanced scalability, resilience, and agility. Using containerization technologies like Docker enables modular deployments. Despite their advantages, effectively evaluating the performance of these dynamic and distributed applications presents challenges. Cloud-native application performance is complex because diverse metrics—including response times, throughput, and resource utilization—are used across varied workloads. Organizations face hurdles in selecting appropriate evaluation tools, defining specific objectives, and gaining insights into microservices and orchestration. A more structured approach is essential to optimizing deployments and meeting performance expectations in cloud-native environments. This paper presents a systematic mapping study that examined 1158 initial papers and refined them to 32 primary studies. It meticulously analyzes key metrics, software tools, objectives, and contributions related to performance evaluation in cloud-native applications. Visual representations with bubble plots were used as cross-correlation analysis, revealing patterns and research gaps. This study provides insights that guide future research directions, advocating for methodological frameworks to enhance the assessment and optimization of cloud-native application performance.

Keywords: mapping study, cloud-native applications, performance, metrics, research contributions

1 Introduction

Cloud-native applications context. The projected growth rate for the global market size of cloud-native applications by 2028 is expected to reach 23.7%, amounting to approximately USD 17 billion [1]. In comparison, the market size stood at USD 5.9 billion in 2023. The popularity of cloud-native applications is illustrated by Google Trends data for the term "cloud-native" over the past 20 years (Figure 1). Cloud-native applications represent a paradigm shift in software development, offering a modern approach to building and deploying applications that are optimized for cloud environments. They are built using a microservices architecture, where applications are decomposed into smaller, loosely coupled services that can be developed, deployed, and scaled independently [3]. Containerization technologies, such as Docker, are often used to package these services into lightweight and portable units, enabling consistent deployment across different environments. Moreover, orchestration tools such as Kubernetes automate container management [12]. Embracing cloud-native principles improves scalability and adaptability [10].

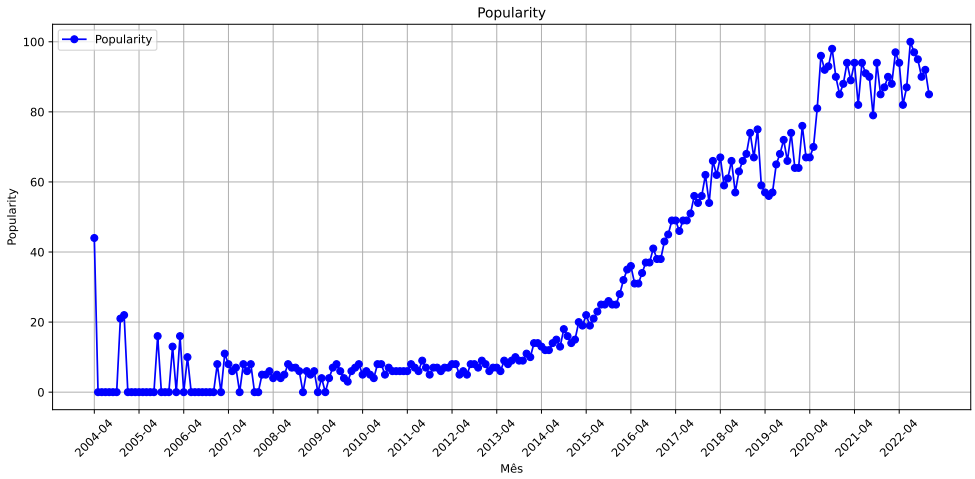


Fig. 1: Google trends regarding cloud-native expression in the last 20 years. Google Trends normalizes search data (0 - 100) to make comparisons between terms easier.

Performance evaluation ensures the effectiveness of cloud-native applications [19]. Cloud-native applications leverage microservices architecture, containerization, and orchestration tools to achieve scalability, resilience, and agility in cloud environments.

However, the dynamic and distributed nature of these applications poses unique challenges for performance assessment [7]. Evaluating the performance of cloud-native applications involves analyzing various metrics such as response times, throughput, resource utilization, and scalability under different workload conditions. The importance of performance evaluation in this context cannot be overstated. It enables organizations to identify bottlenecks, optimize resource allocation, and ensure reliable and responsive user experiences [2]. Moreover, practical performance evaluation facilitates informed decision-making regarding infrastructure provisioning, cost optimization, and application design enhancements. As organizations increasingly rely on cloud-native architectures to meet the demands of modern digital businesses, understanding and optimizing the performance of these applications is essential for maintaining competitiveness and meeting customer expectations.

The research problem. In the context of cloud-native applications, conducting effective performance evaluations poses a challenge. This complexity arises from several factors, including the need to choose the right tools from a wide range of options, define clear objectives for each application, select relevant metrics from numerous possibilities, and determine the potential insights gained from the evaluation process. This challenge is exacerbated by the intricate nature of cloud-native applications, which are based on microservices, containerization, and orchestration. As organizations strive to leverage the scalability, resilience, and agility offered by cloud-native architectures, the lack of a structured approach to performance evaluation hinders their ability to optimize and assess their deployments. This research problem highlights the urgent need for a systematic mapping study to consolidate existing knowledge, identify aspects of performance evaluation, and suggest directions for future research and practical application.

The proposed solution. We conducted a systematic mapping study to explore metrics, software programs, objectives, and contributions related to the performance evaluation of cloud-native applications. We aimed to provide a thorough overview of the current research landscape in this area. Starting with 1158 articles and narrowing down through successive refinement stages to 32 primary studies for detailed analysis. We meticulously analyzed these studies, examining individual factors and their correlations. Utilizing bubble plots, we visually cross-correlated the results, enriching our analysis by illustrating relationships among different factors identified in the literature. This visual method not only facilitates the identification of patterns and trends but also provides deeper insights into the interactions within various aspects of cloud-native application performance evaluation. By integrating quantitative data with visual representations, our study improves understanding and offers a nuanced perspective on the evolving research landscape of cloud-native applications.

The subsequent sections are organized as follows. Section 2 delves into related works. Section 3 describes the methodology used. Section 4 presents classification-based findings. Section 5 discusses a categorization derived from intersecting classifications. Section 6 outlines identified challenges and future directions. Lastly, Section 7 concludes with recommendations for future research in this area.

2 Related Works

This section presents studies that have conducted work similar to ours. We identified thirteen articles that have performed literature reviews, explicitly focusing on systematic mapping studies, systematic literature reviews, or surveys. These reviews centered around various aspects related to cloud-native applications. Table 1 provides a overview of related works. We will discuss the table's contents by reviewing the primary columns sequentially from left to right.

Table 1: Secondary studies that focus on investigating literature regarding aspects of cloud-native applications.

Ref.	Year	Type	Context	Title
[10]	2017	Systematic Mapping Study	General	Understanding cloud-native applications after 10 years of cloud computing-a systematic mapping study
[13]	2019	Systematic Mapping Study	Architecture	A Systematic Mapping Study of Cloud-native Application Design and Engineering
[18]	2021	Survey	Network	Cloud-native network slicing using software-defined networking based multi-access edge computing: A survey
[14]	2022	Survey	Monetary Cost	A Survey on Billing Models for Cloud-Native Applications
[3]	2023	Systematic Literature Review	Architecture	Understanding the challenges and novel architectural models of multi-cloud native applications–a systematic literature review
[9]	2023	Systematic Mapping Study	Architecture	Toward the Observability of Cloud-Native Applications: The Overview of the State-of-the-Art
[16]	2023	Systematic Mapping Study	Security	Static-Analysis-Based Solutions to Security Challenges in Cloud-Native Systems: Systematic Mapping Study
[20]	2023	Survey	Security	Security in Cloud-Native Services: A Survey
[11]	2023	Systematic Mapping Study	Architecture	Multi-tenancy in Cloud-native Architecture: A Systematic Mapping Study
[21]	2023	Survey	Deployment	Cloud-native workload orchestration at the edge: A deployment review and future directions
[17]	2023	Systematic Mapping Study	Security	Access control design practice and solutions in cloud-native architecture: A systematic mapping study
[6]	2023	Survey	Network	A Survey on Resource Management for Cloud Native Mobile Computing: Opportunities and Challenges
[5]	2024	Survey	Architecture	Cloud-Native Computing A Survey From the Perspective of Services
This Paper	**-**	**Systematic Mapping Study**	**Performance Evaluation**	**Performance Evaluation of Cloud Native Applications: A Systematic Mapping Study**

Year of Publication

Analysis of the publication year reveals a trend: the majority of studies emerged after the onset of the pandemic in 2020, with only two papers published prior to

this period, indicating a current and growing interest in research on cloud-native applications. Specifically, eight out of thirteen papers were published in 2023, signaling a marked increase in scholarly attention to this topic during that year. Furthermore, the publication of one paper in 2024 underscores sustained academic interest in the subject.

Type of Research

Systematic mapping studies, *systematic literature reviews*, and *surveys* are methodologies commonly utilized in academic research to consolidate and analyze existing knowledge within a specific domain. A *systematic mapping study* categorizes and maps the breadth of research within a field, aiming to identify trends, gaps, and the scope of available literature. It provides a broad overview by synthesizing a wide range of studies and organizing them based on predefined criteria. In contrast, a *systematic literature review* focuses on synthesizing and analyzing in-depth findings from selected studies, with the objective of providing a deeper understanding of specific research questions or themes. It evaluates methodologies, findings, and theoretical contributions to uncover patterns or contradictions across the literature. *Surveys*, such as narrative reviews or scoping reviews, gather insights from existing literature in a less structured manner. They aim to provide a broad understanding of a topic by summarizing key findings and highlighting trends or issues. Each methodology contributes uniquely to advancing knowledge within their respective fields, collectively enhancing the understanding and applicability of research findings.

We have adopted a systematic mapping study because, in the context of performance evaluation, where definitions, metrics, and methodologies can vary widely, a systematic mapping study ensures that all relevant studies are considered, providing a holistic view of the current state of knowledge. This approach systematically identifies and organizes existing research, clarifying what has been studied and revealing gaps and inconsistencies in the literature. Moreover, this study's systematic design enhances transparency, reproducibility, and the reliability of findings, making it an ideal method for navigating the complexities and uncertainties inherent in evaluating cloud-native applications' performance.

Context

Most of the papers were classified into three categories: architecture, security, and network. We will now discuss these categories specifically.

Architecture. Five recent papers have contributed to understanding various aspects of cloud-native applications in terms of architecture [3, 5, 9, 11, 13]. Deng et al. [5] conducted a survey focusing on issues throughout the cloud-native application lifecycle from a service perspective. Alonso et al. [3] characterized the concept of multi-cloud and defined the characteristics of multi-cloud native applications while also analyzing the current trends and challenges in multi-cloud environments. Odun et al. [13] conducted a systematic mapping study to explore the designs of cloud-native applications and shed light on their frequency in research. Kosinska et al. [9] assessed observability in cloud-native applications through a systematic mapping study, identifying motivations, research areas, and future trends in this domain. Matthew et al.

[11] identified trends, challenges, and opportunities related to multi-tenancy within cloud-native architecture, exploring both improvements and challenges in this area.

Security. Three papers have explored critical aspects of security and access control in cloud-native environments [16, 17, 20]. Theodor et al. [20] explored various security facets specific to cloud-native services, identifying key features and proposing corresponding cybersecurity solutions. Rahaman-a et al. [16] conducted a structured survey focusing on defense mechanisms employed in cloud-native systems, providing insights into the diverse approaches to enhancing system security. Similarly, Rahaman-b et al. [17] analyzed existing access control mechanisms within cloud-native architecture, categorizing access control patterns and offering practical guidance for developers.

Network. Two papers have addressed the network category [6, 18]. Shah et al. [18] investigated network slicing, emphasizing their application in 5G services and exploring recent technological advancements that enable end-to-end (E2E) network slicing. Similarly, Huang et al. [6] conducted a survey on resource management within cloud-native mobile computing, with a specific focus on challenges related to network slicing and containerization resource management.

Others. The remaining three papers were classified into other particular contexts [10, 14, 21]. Vano et al. [21] reviewed the orchestration of cloud-native workloads at the edge, emphasizing challenges related to containerization, exploring pertinent tools, and outlining future directions in edge computing. Meanwhile, Kratzke et al. [10] contributed general insights by defining cloud-native applications through systematic mapping outcomes while also summarizing current research trends and engineering approaches within this domain. Furthermore, Paredes et al. [14] explored micro-billing models for Cloud-Native Applications, identifying potential opportunities and proposing a theoretical framework to enhance billing efficiency in cloud-native environments.

None of the reviewed papers focused on the performance aspect, highlighting a gap in the existing literature. As a result, our study addresses distinct research questions from those explored in prior work. Specifically, we conducted a systematic mapping study to map metrics, tools, objectives, and contributions pertaining to the performance evaluation of cloud-native applications. This approach aims to provide a overview of the current state of research in this area.

3 Methodology

This section presents the methodology of this work. A systematic mapping study was applied to understand the topic of interest. A systematic mapping study performs "*a broad review of primary studies in a specific topic area that aims to identify what evidence is available on the topic.*" [8]. Systematic mapping studies offer several advantages, including providing a overview of a research area, identifying research gaps, and facilitating evidence-based decision-making. By systematically mapping existing literature, researchers can gain insights into the current state of knowledge, highlight areas that require further investigation, and inform future research directions. These studies help synthesize a large volume of information, enabling researchers to identify trends,

patterns, and inconsistencies across studies. This is valuable for policymakers, practitioners, and researchers alike. Additionally, systematic mapping studies help reduce duplication of research efforts, enhance resource allocation efficiency, and promote collaboration among researchers in similar domains. We have adopted the systematic mapping study process proposed by Peterson *et al.* [15]. This process is visualized in Figure 2 and explained in the following subsections.

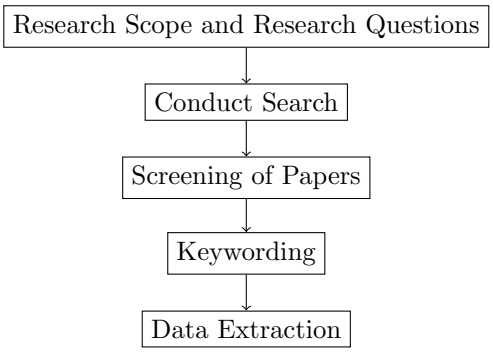


Fig. 2: Steps of the systematic mapping study conducted in this research.

3.1 Research scope and research questions

Understanding the methods for evaluating the performance of cloud-native applications is crucial in today's digital landscape, characterized by migration to cloud-based infrastructure. As organizations increasingly rely on cloud-native architectures to deploy and scale their applications, assessing their performance becomes paramount. The rapid evolution of cloud technologies introduces complexities in performance evaluation, prompting researchers and practitioners alike to explore novel methodologies and tools to address challenges effectively. By delving into the current state-of-the-art, we can uncover approaches, metrics, and frameworks employed to evaluate the performance of cloud-native applications. This exploration observes existing methodologies and serves as a foundation for identifying gaps in the literature, paving the way for future research directions aimed at enhancing performance evaluation practices in cloud-native computing. Based on these arguments, this paper tries to answer the following question:

What is the state-of-the-art related to performance evaluation of cloud-native applications?

Many questions can be derived from such a general question. However, based on the expectations of the specialists and practitioners consulted, we have delimited our efforts to answer four questions presented in Table 2.

RQ01. The state-of-the-art in performance evaluation of cloud-native applications involves assessing various metrics to ensure optimal functionality. Metrics such

Table 2: List of research questions.

Question IDs	Description
RQ01	What are the **metrics** evaluated in this topic?
RQ02	What main **software programs** were used or evaluated in this topic?
RQ03	What are the specific paper **objectives**?
RQ04	What are the **main research contributions**?

as response time, resource utilization, service time, and throughput are essential in evaluating the performance of cloud-native systems [4, 22]. These metrics help in understanding the efficiency, reliability, and scalability of cloud-native applications, enabling researchers and practitioners to identify areas for improvement and optimization. By analyzing these metrics, it becomes possible to enhance the overall performance of cloud-native systems, ensuring they meet the required service level objectives and deliver a seamless user experience. However, it is interesting to know how the majority of metrics are used in this area and how they can be combined with other aspects.

RQ02. Understanding the primary software programs employed or assessed in the performance evaluation of cloud-native applications holds importance for various reasons.

The software Kubernetes and AWS tools are highly used tools in this area. However, there are many others. This question provides invaluable insights into the technological ecosystem of cloud-native environments, unveiling the predominant tools and platforms utilized by both practitioners and researchers. This understanding empowers stakeholders to make informed decisions regarding tool selection and infrastructure choices for their performance evaluation endeavors. Identifying the leading software programs offers a nuanced understanding of their capabilities, limitations, and compatibility with diverse cloud-native architectures. Such knowledge facilitates the selection of appropriate tools tailored to specific evaluation requirements, ensuring optimal performance assessment outcomes. This exploration not only fosters continuous improvement in evaluation methodologies but also drives the development of more sophisticated and practical tools to address evolving performance challenges in cloud-native environments.

RQ03. Understanding the specific objective of cloud-native performance evaluation is crucial for several reasons. Firstly, cloud-native architectures introduce a myriad of unique objectives, such as orchestration, resource management, optimization, and benchmarking. Focusing on specific objectives allows researchers and practitioners to tailor their evaluation methodologies to address these nuances effectively. Pinpointing and defining these specific elements provide emphasis and clarity to performance evaluation processes. Exploring narrowly targeted areas promotes a deeper understanding of the complexities of cloud-native settings. This greater understanding may result in the creation of specialized instruments, measurements, and benchmarks precisely calibrated to evaluate performance in these particular areas.

RQ04. Understanding the main research contributions in cloud-native performance evaluation, advances our understanding and capabilities in optimizing cloud-native applications. By identifying key areas such as evaluation, method, tool, and algorithm,

we uncover the diverse facets of performance assessment within cloud-native environments. Highlighting the leading research contributions in cloud-native performance evaluation underscores the importance of ongoing research in this area. Exploring these contributions allows us to identify trends, address existing challenges, and propel the field forward, ultimately improving the performance and scalability of cloud-native applications in real-world scenarios.

3.2 Conduct search (for primary studies)

We conducted our search for studies in the electronic sources listed in Table 3. We searched for contributions published with the three terms "cloud," "native," and "performance" in their title, abstract, or keywords. However, we did not perform a full-text search. Authors should intentionally consider the cloud-native performance evaluation aspect so much that they explicitly mention it in the title, abstract, or as a keyword.

Table 3: Search strings specific for each search engine.

Search Engines	Quantity	Search String
IEEE Xplore	422	("Abstract":cloud native performance)
ACM Digital Library	192	[Abstract: cloud] AND [Abstract: native] AND [Abstract: performance]
Google Scholar	355	Search for "cloud native performance", then click on Sort by date - Abstracts
ScienceDirect	75	Title, abstract, keywords: cloud AND native AND performance
SpringerLink	114	cloud AND native AND performance
TOTAL	**1158**	

3.3 Screening of papers (for inclusion and exclusion)

Inclusion and exclusion criteria are indispensable elements of mapping studies, serving as a navigational compass that guides researchers through the vast sea of literature. These criteria ensure precision in research scope by delineating the boundaries of relevance, allowing scholars to focus exclusively on literature that directly addresses their research questions and objectives. They minimize bias by providing transparent guidelines for selection and optimize resource utilization by efficiently allocating time and effort to synthesizing high-quality evidence. Moreover, adherence to these criteria improves the reproducibility and transparency of mapping studies, enabling other researchers to validate findings and build upon existing knowledge. In essence, inclusion and exclusion criteria are the cornerstones of methodological rigor in mapping studies. They facilitate the synthesis of meaningful insights that contribute to the advancement of scholarly discourse and understanding in various fields. Table 4 presents the inclusion and exclusion criteria for this work. Table 5 shows the 32 selected studies.

Table 4: Inclusion and exclusion criteria.

Inclusion	Exclusion
• The abstract explicitly mentions the performance evaluation of cloud-native applications. • From the abstract, we can deduce that the focus of the paper contributes to the chosen research focus. • Abstract/keywords include key terms, and it is clear from the abstract that a contribution towards cloud-native applications and their evaluation is made.	• The paper lies outside the performance evaluation of cloud-native applications context. • Performance evaluation of cloud-native applications is not part of the contributions of the paper; the terms are only mentioned in the general introductory sentences of the abstract. • Literature not peer-reviewed and only in the form of abstract, blog, or presentation are excluded.

Table 5: Primary studies detailed as references.

ID	Title
PS01	"Performance evaluation of two service recovery strategies in cloud-native radio access networks," Shunmugapriya Ramanathan et al., 2019 21st International Conference on Transparent Optical Networks (ICTON), IEEE, 2019.
PS02	"Towards latency-sensitive cloud-native applications: A performance study on AWS," István Pelle et al., 2019 IEEE 12th International Conference on Cloud Computing (CLOUD), IEEE, 2019.
PS03	"X-containers: Breaking down barriers to improve performance and isolation of cloud-native containers," Zhiming Shen et al., Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, 2019.
PS04	"Control-over-the-cloud: A performance study for cloud-native, critical control systems," Per Skarin et al., 2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC), IEEE, 2020.
PS05	"Evaluations of network performance enhancement on cloud-native network function," Yong-Xuan Huang, Jerry Chou, Proceedings of the 2021 on Systems and Network Telemetry and Analytics, 2020, 3-8.
PS06	"PRESTO: a latency-aware power-capping orchestrator for cloud-native microservices," Rolando Brondolin, Marco D. Santambrogio, 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS), IEEE, 2020.
PS08	"Validation and Benchmarking of CNFs in OSM for pure Cloud Native applications in 5G and beyond," Adrián Pino et al., 2021 International Conference on Computer Communications and Networks (ICCCN), IEEE, 2021.
PS09	"CloudNPlay: Resource optimization for a cloud-native gaming system," Angelus Wibowo, Ta Nguyen Binh Duong, 2021 IEEE 30th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), IEEE, 2021.
PS10	"High-performance software load balancer for cloud-native architecture," Jung-Bok Lee et al., IEEE Access, v. 9, p. 123704-123716, 2021.
PS11	"Performance evaluation and improvement of a cloud-native data analysis system application," Chia-Chuan Chuang, Yu-Chin Tsai, 2021 international conference on electronic communications, internet of things and big data (ICEIB), IEEE, 2021.
PS07	"SRv6-PM: A cloud-native architecture for performance monitoring of SRv6 networks," Pierpaolo Loreti et al., IEEE Transactions on Network and Service Management, v. 18, n. 1, p. 611-626, 2021.
PS12	"A configurable method for benchmarking scalability of cloud-native applications," Sören Henning, Wilhelm Hasselbring, Empirical Software Engineering, v. 27, n. 6, p. 143, 2022.
PS13	"Applicability of cloud native-based healthcare monitoring platform (cn-hmp) in older adult facilities," Abu Bony Amin et al., 2022 44th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC), IEEE, 2022.

Continued on next page

Table 5 – continued from previous page

ID	Title
PS14	"Availability and latency aware deployment of cloud native edge slices," Sagar Arora, Adlen Ksentini, Christian Bonnet, GLOBECOM 2022-2022 IEEE Global Communications Conference, IEEE, 2022.
PS15	"Democratizing the computational environmental marine data science: using the high-performance cloud-native computing for inert transport and diffusion lagrangian modelling," Gennaro Mellone et al., 2022 IEEE International Workshop on Metrology for the Sea; Learning to Measure Sea Health Parameters (MetroSea), IEEE, 2022.
PS16	"Latency and network aware placement for cloud-native 5G/6G services," Kiranpreet Kaur et al., 2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC), IEEE, 2022.
PS17	"Microservices configurations and the impact on the performance in cloud-native environments," Mohamed Mekki, Nassima Toumi, Adlen Ksentini, 2022 IEEE 47th Conference on Local Computer Networks (LCN), IEEE, 2022.
PS18	"Novel Abstraction and Offload Mechanisms for High Performance Cloud-native Distributed Object Stores," Anoop Reddy et al., 2022 IEEE International Conference on Cloud Engineering (IC2E), IEEE, 2022.
PS19	"Optimizing Performance and Resource Consumption of Cloud-Native Logging Application Stacks," Gergő Csáti, István Pelle, László Toka, NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium, IEEE, 2022.
PS20	"Performance Evaluation of Resource Management Schemes for Cloud Native Platforms with Computing Containers," Yuqi Fu et al., 2022 IEEE International Performance, Computing, and Communications Conference (IPCCC), IEEE, 2022.
PS21	"Scalable Digital Pathology Platform Over Standard Cloud Native Technologies," Tibério Baptista et al., 2022 IEEE Symposium on Computers and Communications (ISCC), IEEE, 2022.
PS22	"Toward Performance Efficient UAV Task Scheduling in Cloud Native Edge," Siqi Huang, Deze Zeng, Zhihao Qu, GLOBECOM 2022-2022 IEEE Global Communications Conference, IEEE, 2022.
PS23	"A Comprehensive Performance Analysis of Stream Processing with Kafka in Cloud Native Deployments for IoT Use-cases," István Pelle et al., NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium, IEEE, 2023.
PS24	"CDSBen: Benchmarking the Performance of Storage Services in Cloud-Native Database System at ByteDance," Jiashu Zhang et al., Proceedings of the VLDB Endowment, v. 16, n. 12, p. 3584-3596, 2023.
PS25	"Cloud-native-bench: an extensible benchmarking framework to streamline cloud," Michiel Van Kenhove et al., Proceedings of IEEE CloudNet2024, the IEEE International Conference on Cloud Networking, 2023.
PS26	"High-Level Data Abstraction and Elastic Data Caching for Data-Intensive AI Applications on Cloud-Native Platforms," Rong Gu et al., IEEE Transactions on Parallel and Distributed Systems, 2023.
PS27	"PerfSim: A performance simulator for cloud-native microservice chains," Michel Gokan Khan et al., IEEE Transactions on Cloud Computing, v. 11, n. 2, p. 1395-1413, 2021.
PS28	"Zero-cost in-depth enforcement of network policies for low-latency cloud-native systems," Gerald Budigiri et al., 2023 IEEE 16th International Conference on Cloud Computing (CLOUD), IEEE, 2023.
PS29	"Disambiguating Performance Anomalies from Workload Changes in Cloud-Native Applications," Alexandru Baluta et al., Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering, 2024.
PS30	"Lightpool: A nvme-of-based high-performance and lightweight storage pool architecture for cloud-native distributed database," Jiexiong Xu et al., 2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA), IEEE, 2024.
PS31	"Performance Optimization Across the Edge-Cloud Continuum: A Multi-agent Rollout Approach for Cloud-Native Application Workload Placement," Polyzois Soumplis et al., SN Computer Science, v. 5, n. 3, p. 318, 2024.

Continued on next page

Table 5 – continued from previous page

ID	Title
PS32	"Towards High-Performance Data Loading in Cloud-Native Deep Learning Systems," Zhuangwei Kang et al., 2024 16th International Conference on COMmunication Systems & NETworkS (COMSNETS), IEEE, 2024.

3.4 Keywording

We have read all the selected papers and extracted the keywords that could answer one of the four research questions. We have found 48 words (or expressions), which were grouped (and ordered by importance) and presented in Table 6. It is essential to highlight that the words were unified when they appeared with similar meanings, as suggested by Peterson et al. [15]. We have found 20 metrics, 17 software programs, 7 objectives, and 4 main research contributions.

Table 6: Unified keywords.

Classification	Keywords
Metrics	latency, response time, service time, throughput, resource utilization, scalability, packet loss, power consumption, monetary cost, availability, number of successful executions, frame rates, instantiation time, Kafka KPIs, IO rate, simulation accuracy, simulation speed, precision, recall, resource allocation success.
Software	Kubernetes, Some AWS tool, Prometheus, Docker, Proprietary tool, Tensorflow, Pytorch, Spark, Grafana, gRPC, Big Data tools, OSM-OpenStack, Kafka, Fluid, Theodolite, PROnet Orchestrator, libOS, Calico.
Objective	orchestration, resource management, optimization, benchmarking, monitoring, storage, network performance.
Main Research Contribution	evaluation, method, tool, algorithm.

3.5 Data extraction

In this step, we associate the keywords to each paper, producing the results listed in Table 7. This table allows the reader to access the specific paper that addresses the desired characteristics. However, we will analyze each category in two ways: isolated and associated with each other.

Table 7: Categorization of each paper.

ID	Metrics	Software	Objective	Main Research Contribution
PS01	throughput and resource utilization	PROnet Orchestrator	orchestration	evaluation
PS02	throughput, service time, number of successful executions	Some AWS tools	orchestration	evaluation
PS03	latency, packet loss and throughput	libOS	optimization	method
PS04	latency and service time	Some AWS tools	benchmarking	evaluation
PS05	latency, packet loss, and throughput	Calico and Kubernetes	network performance	evaluation

Continued on next page

Table 7 – continued from previous page

ID	Metrics	Software	Objective	Main Research Contribution
PS06	power consumption, packet loss and latency	Docker and Kubernetes	orchestration	tool
PS08	latency, frame rates, response time, monetary cost	Some AWS tools	resource management	tool
PS09	throughput	Kubernetes	orchestration	evaluation
PS10	resource utilization and response time	Kubernetes, and Docker	resource management	evaluation
PS11	throughput	gRPC and Big Data tools	monitoring	method
PS07	latency and power consumption, resource utilization, instantiation time	Kubernetes and OSM-OpenStack	benchmarking	evaluation
PS12	scalability	Kubernetes	benchmarking	method
PS13	response time, packet loss, and latency	Proprietary tool	monitoring	evaluation
PS14	availability, latency	Kubernetes	orchestration	evaluation
PS15	response time, throughput, service time, scalability, and resource utilization	Kubernetes and Prometheus	orchestration	method
PS16	latency, service time	Kubernetes	optimization	algorithm
PS17	service time	Kubernetes	resource management	evaluation
PS18	response time, throughput, service time, scalability, and resource utilization	Some AWS tools	storage	evaluation
PS19	response time, throughput, and scalability	Kubernetes and Some AWS tools	monitoring	evaluation
PS20	service time, resource utilization and response time	Some AWS tools, Docker, Kubernetes, Tensorflow, Pytorch, and Spark	resource management	evaluation
PS21	response time and throughput	Kubernetes	resource management	method
PS22	service time, availability and latency	Some AWS tools	optimization	algorithm
PS23	response time and Kafka KPIs	Kafka	benchmarking	evaluation
PS24	latency, IO rate	Proprietary tool	storage	tool
PS25	scalability	Kubernetes	benchmarking	tool
PS26	response time and throughput	Fluid	orchestration	method
PS27	simulation accuracy, simulation speed, response time	Kubernetes	orchestration	tool
PS28	latency, throughput, response time, resource utilization	Kubernetes	resource management	algorithm
PS29	precision, recall	Theodolite and Kubernetes	resource management	method
PS30	resource utilization, latency, and throughput	Kubernetes	storage	method

Continued on next page

Table 7 – continued from previous page

ID	Metrics	Software	Objective	Main Research Contribution
PS31	resource allocation success, service time, latency, resource utilization, monetary cost	Proprietary tool	optimization	algorithm
PS32	throughput	Kubernetes	optimization	method

4 Research Answers Analyzed Individually

In this section, we examine each question's response individually, aiming to extract the most and least utilized options while providing insightful commentary. We visualize the frequency of occurrence for each option using bar plots, facilitating clear comparisons. Throughout our analysis, we focus on discussing the most adopted items, providing insight into their relevance within the study's context.

4.1 RQ01 - Metrics

Figure 3 shows the used metrics organized by number of occurrences. The first six metrics are intrinsically related to performance, and therefore, we focus on discussing them. The three most used metrics (latency, response time, and service time) can be grouped as measures related to time. These metrics are correlated metrics that collectively provide insight into the performance of systems and networks. Latency represents the time taken for data to traverse between endpoints. Response time encompasses latency along with the time spent waiting in queues and processing the request on the server. Within response time, service time isolates the duration spent exclusively on processing the request by the server, excluding network-related delays. These three metrics offer a view of system and network performance and are easily perceived by the end user [PS08] [PS01] [PS04].

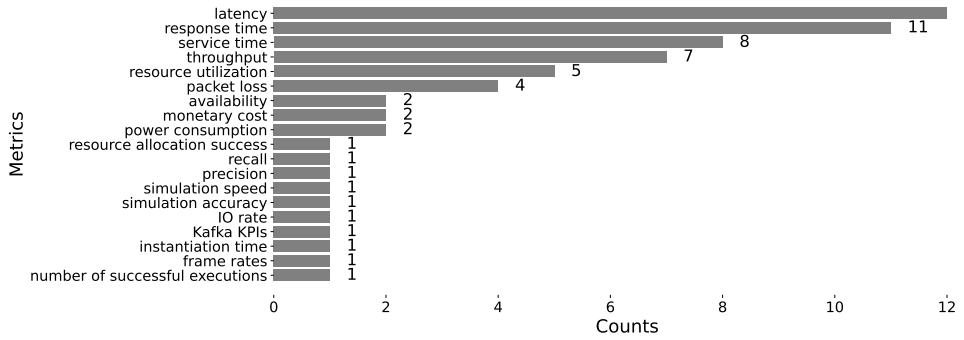


Fig. 3: Primary studies quantified by metrics

Throughput is the fourth metric and gives a more holistic view of infrastructure performance. It provides information on the application's ability to handle

increasing workloads, ensuring optimal resource utilization (the fifth metric) while maintaining responsiveness. Moreover, in the context of microservices, throughput measurements aid in identifying performance bottlenecks and optimizing service interactions. Throughput performance was extensively analyzed in the research papers provided. For instance, the papers discussed how increasing computation performance benefits accessing the same key with a few concurrent functions [PS02], the performance differences between loopback and Load Balancer Node (LBN) in terms of bps rate and pps rate [PS09], the system call throughput improvements with X-Containers compared to Docker and Clear Containers [PS03], the impact of gradient computation time on throughput for ImageNet and LibriSpeech datasets [PS32], and the mean throughput of different web servers under specific resource restrictions [PS25]. The sixth metric (packet loss) is primarily used when the evaluator observes failure aspects, mainly during stress tests. Packet loss is sometimes hard to reach, considering powerful computing equipment, but it was observed in four papers.

4.2 RQ02 - Software Programs

Figure 4 shows the used software programs organized by the number of occurrences. Kubernetes was extensively utilized in the research papers studied. The studies implemented Kubernetes for various purposes, such as managing applications in containers, conducting performance analyses, and automating benchmark deployments. For instance, Kristiani et al. [PS02] utilized Kubernetes to implement an edge-computing architecture and distributed cloud applications. Additionally, the studies highlighted the importance of Kubernetes operators like Theodolite to benchmark scalability in microservices and ImageJockey to evaluate container image performance [PS09]. Furthermore, the research emphasized the significance of Kubernetes in orchestrating cloud-native applications, enforcing resource requests and limits, and effectively managing resource consumption [PS05]. In summary, Kubernetes facilitates efficient cloud architectures, automating benchmarking processes, and improving the performance of containerized applications across various studies.

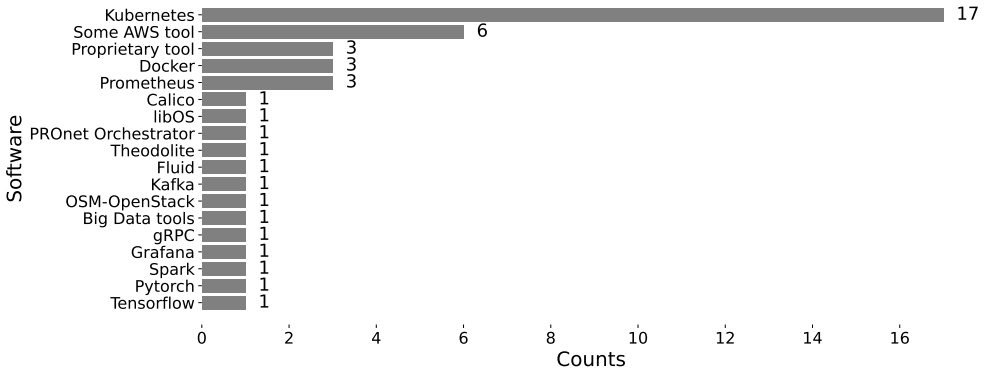


Fig. 4: Primary studies quantified by software programs.

In the second position resides Some AWS tools. AWS tools were extensively utilized for various purposes. The studies involved performance benchmarking of AWS Lambda and ECS across different regions, using tools like CloudWatch Logs to monitor hardware configurations [PS02]. Additionally, the papers discussed the implementation of a cloud-native deep learning system that leverages AWS S3 for data storage and access, showcasing the integration of AWS services within the system [PS27]. The analysis of cloud-native architectures and load-balancing mechanisms highlighted the importance of AWS tools in optimizing performance and scalability within cloud environments [PS09] [PS32]. Overall, the research papers demonstrated the diverse applications and benefits of utilizing AWS tools in enhancing cloud services and system functionalities. Three papers proposed proprietary tools, with Docker being utilized in only three papers despite its use in microservices contexts. Prometheus is featured prominently in research papers and has been extensively applied for diverse monitoring and management purposes. In one study [PS21], Prometheus was integrated into the proposed architecture using the Prometheus Stack Helm Package, connecting with the K8s' API and producing metrics about nodes and objects in the K8S environment. Additionally, Prometheus was leveraged as an open-source monitoring solution, integrating with multiple metrics exporters to monitor applications and their environments alongside application metrics averages output in httperf logs [PS29]. Additionally, another paper [PS25] emphasized the extension of the metrics server through the Prometheus Adapter Helm Package component. This enhancement enables Prometheus to query the deployment for custom metrics, facilitating precise system scaling based on metrics like concurrent DICOM associations and process memory usage.

4.3 RQ03 - Objectives

Figure 5 shows the objectives organized by number of occurrences. Resource management and orchestration are important in the mapping study on cloud-native performance evaluation due to their fundamental roles in ensuring the efficient operation and scalability of cloud-native applications. With the dynamic nature of cloud environments and the proliferation of microservices architectures, effective resource management is essential for optimizing resource allocation, maximizing utilization, and minimizing costs. Similarly, orchestration plays a critical role in automating the deployment, scaling, and management of application components, enabling coordination and efficient resource utilization across distributed environments. Container orchestrators like Kubernetes play a vital role in automating the deployment and management of containerized components within clusters, enabling features such as automatic container placement, replication, failover recovery, and scaling [PS21] [PS09]. Additionally, optimizing application performance while managing power consumption is essential, especially considering the non-linear relationship between server performance and energy usage, emphasizing the need for power-capping orchestrators to maintain workload performance near predefined SLAs [PS06]. Effective resource management and orchestration not only enhance software reliability and availability but also enable the creation, removal, and scaling of applications to meet varying traffic demands, ultimately ensuring the reliable and scalable delivery of cloud services [PS20].

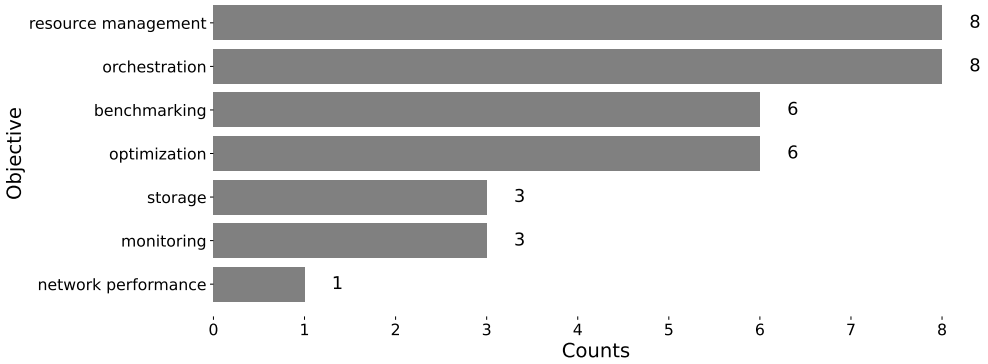


Fig. 5: Primary studies quantified by objectives.

Benchmarking also stands out as the primary objective. Benchmarking allows for the comparison of different configurations, architectures, and technologies, providing valuable insights into performance variations and identifying best practices. By benchmarking cloud-native applications, researchers can assess their performance under various conditions and workload scenarios, facilitating informed decision-making and optimization efforts. Benchmarking in empirical software engineering serves to compare and rank different software under test, evaluate new methods or tools against established standards, and repeat previous experiments [PS12]. The complexity of cloud environments, with various storage options and cache systems, highlights the need for benchmarking to ensure optimal performance in data-intensive applications like deep learning training jobs [PS26]. Automated benchmarking tools like Cloud-Native-Bench streamline the process by monitoring benchmark states, automating consecutive executions, and facilitating result collection and analysis, enhancing the ease of use and efficiency of benchmarking workflows [PS25]. Benchmarking not only aids in understanding the scalability and performance of distributed systems but also enables users to make informed decisions based on specific business needs, ensuring the practical evaluation of different technologies in cloud environments.

Furthermore, optimization emerges as a central objective. In the context of cloud-native applications, optimization refers to the systematic process of fine-tuning various aspects of application design, infrastructure configuration, and resource utilization to improve performance, scalability, efficiency, and cost-effectiveness. Optimization involves identifying and addressing bottlenecks, inefficiencies, and areas for improvement within the application architecture and deployment environment. Optimization efforts include optimizing resource allocation and utilization, optimizing data storage and retrieval mechanisms, tuning networking configurations, and optimizing application code and algorithms. The goal of optimization in cloud-native applications is to maximize the utilization of available resources, minimize response times, enhance scalability, and ultimately improve the overall quality of service provided to users while minimizing operational costs. For instance, optimizing performance and resource utilization during deep learning training is essential, given the complexity of managing

diverse storage systems and cache configurations in cloud environments [PS19] [PS26]. In this case, the motivation for optimization stems from the need to efficiently handle massive datasets and specialized computing accelerators in deep learning workloads, which are both data-hungry and computation-intensive.

4.4 RQ04 - Main Research Contributions

Figure 6 shows the main research contributions organized by a number of occurrences. Performance evaluation serves as the cornerstone contribution in the mapping study on cloud-native performance evaluation due to its inherent significance in assessing the effectiveness, reliability, and scalability of cloud-native applications. The mapping study focuses on performance evaluation and aims to provide a overview of existing methodologies, metrics, and tools for assessing the performance of cloud-native applications. The majority of the papers focused on performance evaluation of existing or proposed platforms. For instance, the evaluation of container platforms like Docker Swarm and Kubernetes on various setups was a vital aspect of the research in [PS20]. Similarly, other papers compared Management and Orchestration (MANO) platforms, such as OSM and ONAP, in terms of operational and functional metrics without introducing new tools [PS07]. Additionally, experimental evaluations were conducted on stream processing engines in Kubernetes clusters across different cloud environments to assess benchmarking results and execution times, emphasizing performance evaluation over tool development [PS10].

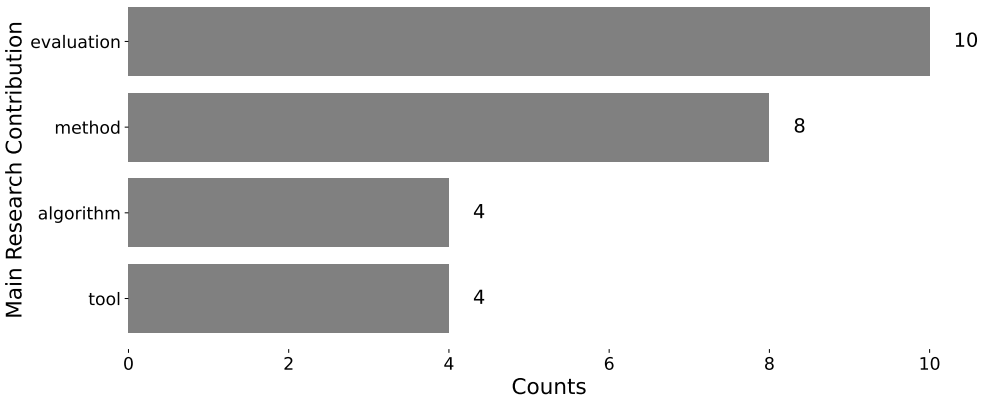


Fig. 6: Primary studies, quantified by main research contribution.

As cloud-native architectures continue to evolve, novel methods are essential for addressing the unique challenges associated with evaluating and optimizing the performance of these complex distributed systems. Several papers have proposed methods and platforms in the realm of cloud-native technologies and data science. These include the introduction of X-Containers and SRv6-PM for efficient deployment and management of distributed applications [PS13] [PS20], as well as the development of a Scalable

Digital Pathology Platform utilizing standard cloud-native technologies to enhance medical imaging practices and enable remote diagnostics and telemedicine [PS21]. Additionally, the democratization of computational environmental marine data science has been addressed, showcasing advances in cloud-based architectures for demanding imaging scenarios with high-resolution images, ensuring scalability, data replication, and auto-scaling mechanisms for improved query throughput and reduced retrieval times [PS12].

5 Research Answers Analyzed Associatively

This section elucidates the frequency of studies based on the interrelationship between two distinct criteria, displaying the number of studies at the intersections of the x and y axes within bubble plots. This multidimensional analysis, considering various criteria simultaneously, helps identify prevalent research domains and potential gaps for future exploration. Therefore, we will focus on these two aspects, drawing red and green rectangles to discussed examples of prevalent research domains and potential gaps, respectively. The color red indicates that the topic may have been overexplored, and green indicates that further research may be necessary.

5.1 Metrics x Main Research Contribution

Figure 7 presents the relationship between the metrics and the main research contribution.

Prevalent Research Domains

The biggest bubble is the combination of throughput with the method. Throughput is closely tied to papers that introduce novel methods for benchmarking scalability in cloud-native applications [PS10][PS12][PS30][PS20]. [PS12] emphasizes the importance of scalability and performance in handling large datasets efficiently. [PS30] focuses on enhancing throughput by leveraging cloud-native technologies for efficient data processing and analysis. [PS20] also contributes to improving throughput by implementing advanced data abstraction and caching mechanisms for AI workloads on cloud-native environments. These papers collectively highlight the importance of methodological advancements in achieving high throughput in cloud-native applications.

Potential Gaps

In the list of metrics discussed earlier, three metrics appeared only twice: packet loss, availability, and monetary cost. Despite their importance, these metrics have yet to be extensively explored. Deducing from the bubble plot, packet loss has a gap in the papers proposing algorithms. Tools like iperf accurately measure packet loss, indicating the reliability of counting systems in monitoring losses [PS11].

Availability offers a gap in method's and tool's papers. Availability is an essential aspect in various domains, such as telecomunication services, cloud-native systems, and object storage solutions. In the context of cloud-native systems, ensuring high

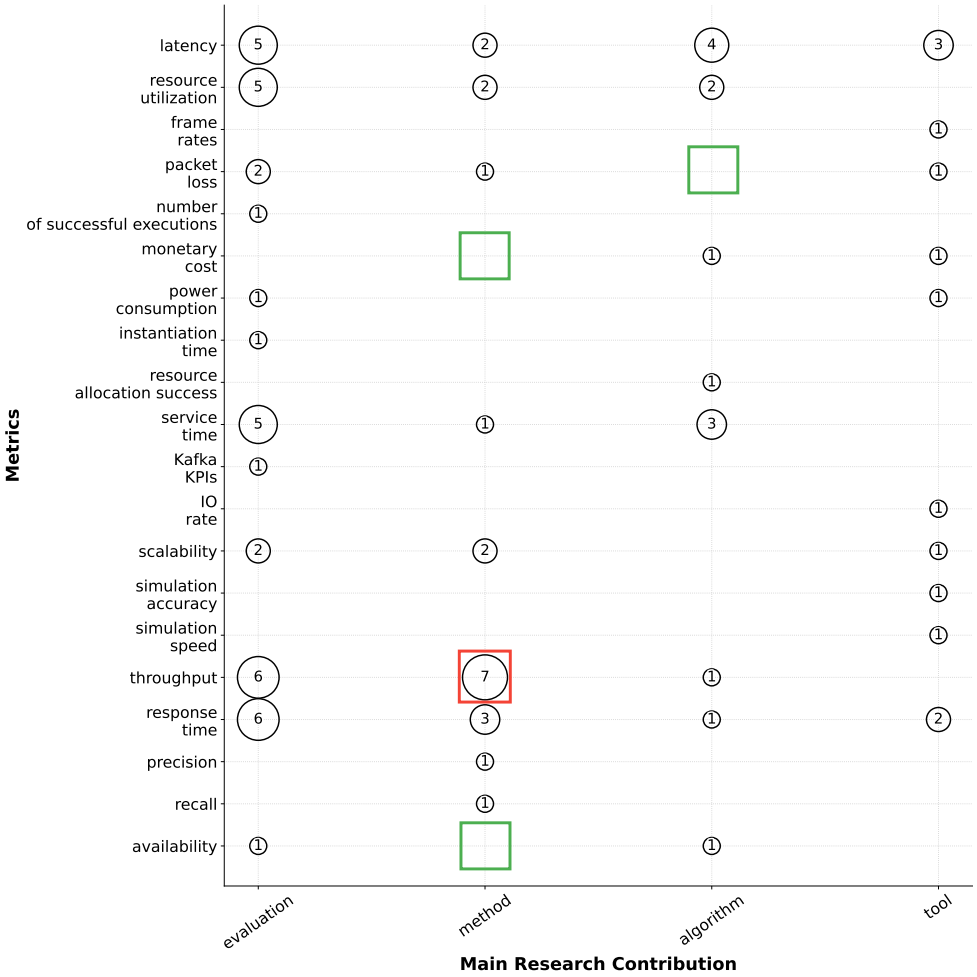


Fig. 7: Combination between metrics and main research contribution.

availability is essential to minimize downtime and guarantee active microservices, with the goal of achieving 99.999% uptime [PS03]. Hot upgrades and hot migration in storage services increase availability, which prevents interruptions during component upgrades and data migration, ensuring continuous service availability [PS30]. Finally, the monetary cost has a gap in evaluation and method papers. When aiming to minimize monetary costs, the preference shifts towards utilizing cost-effective cloud nodes. Networking costs contribute to the total cost, especially when transferring data to the cloud. In contrast, prioritizing latency minimization through edge resources results in the processing cost of edge resources being the primary factor influencing the total monetary cost, with networking costs playing a minor role [PS31].

5.2 Software x Main Research Contribution

Figure 8 presents the relationship between software and main research contribution.

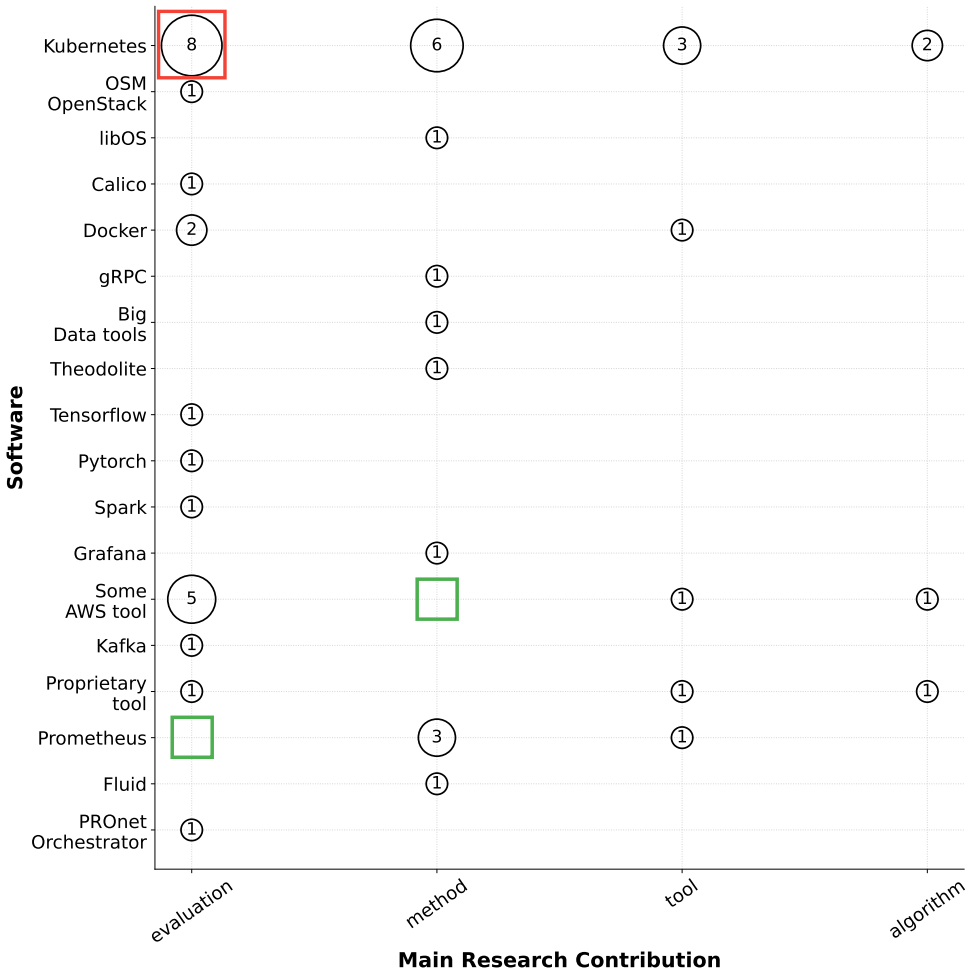


Fig. 8: Combination between software and contribution.

Prevalent Research Domains

Among the many research efforts, eight papers illuminate the dynamics of application performance evaluation in Kubernetes-based environments. These studies explore various facets, ranging from container orchestration strategies to resource management techniques, all aimed at optimizing efficiency and scalability in cloud-native

ecosystems. Through experimentation and analysis, these papers contribute to the collective understanding of how Kubernetes functions as a framework in modern cloud computing landscapes. Kubernetes is extensively utilized in performance evaluation studies, as evidenced by various research papers. The evaluation of resource management schemes for cloud-native platforms with computing containers highlights the importance of Kubernetes in optimizing performance [PS20]. Furthermore, the study on optimizing performance and resource consumption of cloud-native logging application stacks showcases how Kubernetes enhances efficiency and scalability in such environments [PS12]. Overall, these papers collectively demonstrate Kubernetes' significance in driving performance improvements and resource optimization in cloud-native applications.

Potential Gaps

There is a notable gap in research concerning the utilization of Prometheus in performance evaluation for cloud-native applications. Although many studies focus on methodologies or tools, more research is needed on leveraging Prometheus for performance assessment. Prometheus, with its monitoring capabilities, offers a promising approach for evaluating and optimizing the performance of cloud-native applications. By harnessing its metrics collection and querying functionalities, researchers could delve deeper into understanding the intricate interplay between application components and Kubernetes infrastructure, thus fostering more informed decision-making processes to enhance performance and scalability.

Prometheus offers numerous advantages in monitoring and managing metrics within various environments. It integrates with different metrics exporters, providing a view of application and system performance [PS29]. By connecting with Kubernetes' API and utilizing components such as node exporter and kube-state-metrics, Prometheus enables detailed monitoring of cluster nodes and Kubernetes objects, enhancing system visibility and control [PS21]. Additionally, Prometheus can be extended to collect custom metrics from deployments, allowing fine-grained control over system scaling and performance optimization [PS25]. Furthermore, by supplementing real-time resource insights and facilitating the integration of tracing tools, Prometheus enhances data analysis capabilities and aids in identifying bottlenecks in large-scale applications. Within the domain of cloud-native applications, a prominent research gap lies in leveraging AWS tools to introduce novel methods. While existing studies have made strides in various aspects, such as tool development or empirical analyses, more methodological papers are still needed to explore the potential of AWS services in advancing cloud-native paradigms. AWS offers a diverse array of services, from container orchestration with Amazon ECS or EKS to serverless computing with AWS Lambda, providing a rich landscape for methodological frameworks. By elucidating new methodologies that harness the capabilities of AWS tools, researchers could unlock fresh avenues for optimizing performance, scalability, and resilience in cloud-native applications.

5.3 Objective x Main Research Contribution

Figure 9 presents the relationship between objective and main research contribution.

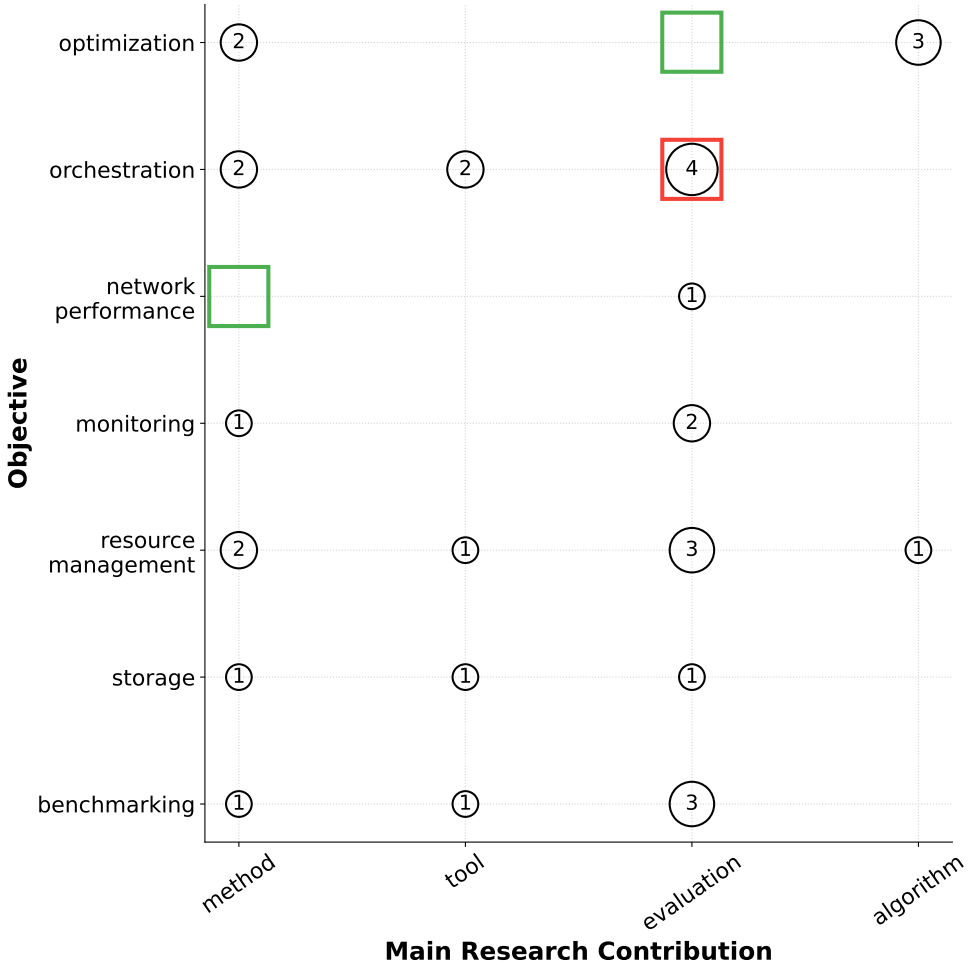


Fig. 9: Combination between objective and contribution.

Prevalent Research Domains

Papers that primarily contribute to performance evaluation often concentrate on orchestration in cloud-native applications. These studies assess the efficacy of various orchestration strategies within cloud-native environments, addressing their impact on performance metrics such as latency, throughput, and resource utilization. By meticulously analyzing orchestration techniques, such as those facilitated by Kubernetes,

Docker Swarm, or Amazon ECS, these papers provide invaluable insights into the optimal deployment and management of containerized applications. As the research papers highlight, orchestration is familiar in cloud-native architectures to ensure high availability and low latency. The deployment of cloud-native edge slices emphasizes the need for efficient resource allocation and management to avoid over-provisioning and optimize performance [PS06]. Additionally, the performance evaluation of service recovery strategies in cloud-native radio access networks underscores the importance of orchestrating recovery mechanisms to maintain network reliability and minimize downtime [PS21]. Furthermore, the implementation of a high-performance software load balancer in cloud-native environments showcases the significance of orchestrating load distribution to enhance system availability and responsiveness [PS17].

Potential Gaps

A notable research gap emerges in the evaluation of network performance within papers where the method serves as the primary contribution. While other types of contributions, such as tool development or empirical studies, have garnered attention, methodological papers specifically addressing network performance evaluation still need to be included. By focusing on this aspect, researchers can offer insights and practical solutions to address the unique challenges of distributed networking in the cloud, ultimately advancing the efficiency, reliability, and scalability of cloud-native applications.

Another noticeable gap exists in research pertaining to optimization within papers where the evaluation itself serves as the primary contribution. While numerous studies focus on evaluating various aspects of cloud-native architectures, ranging from performance to scalability, more research still needs to be dedicated to optimization methodologies within this context. This gap presents an opportunity for exploration aimed at developing novel optimization techniques explicitly tailored to the intricacies of cloud-native environments. By addressing this gap, researchers can contribute insights and practical solutions to enhance the efficiency, resource utilization, and overall performance of cloud-native applications. Closing this research gap holds immense promise for driving advances in cloud-native computing.

5.4 Metrics x Objectives

Figure 10 presents the relationship between metrics and objectives.

Prevalent Research Domains

Latency has been extensively explored across various objectives, with a predominant focus on optimization within the evaluation process. From enhancing performance to fine-tuning resource allocation, researchers have scrutinized latency metrics to glean insights into optimizing cloud-native applications for efficiency and responsiveness. By exploring latency, studies aim to streamline operations, minimize response times, and maximize user experience in distributed environments. This exploration underscores the critical role of latency optimization in the continual improvement of cloud-native ecosystems. Utilizing the latency metric is motivated by its critical role in driving the deployment of cloud-native microservices at the

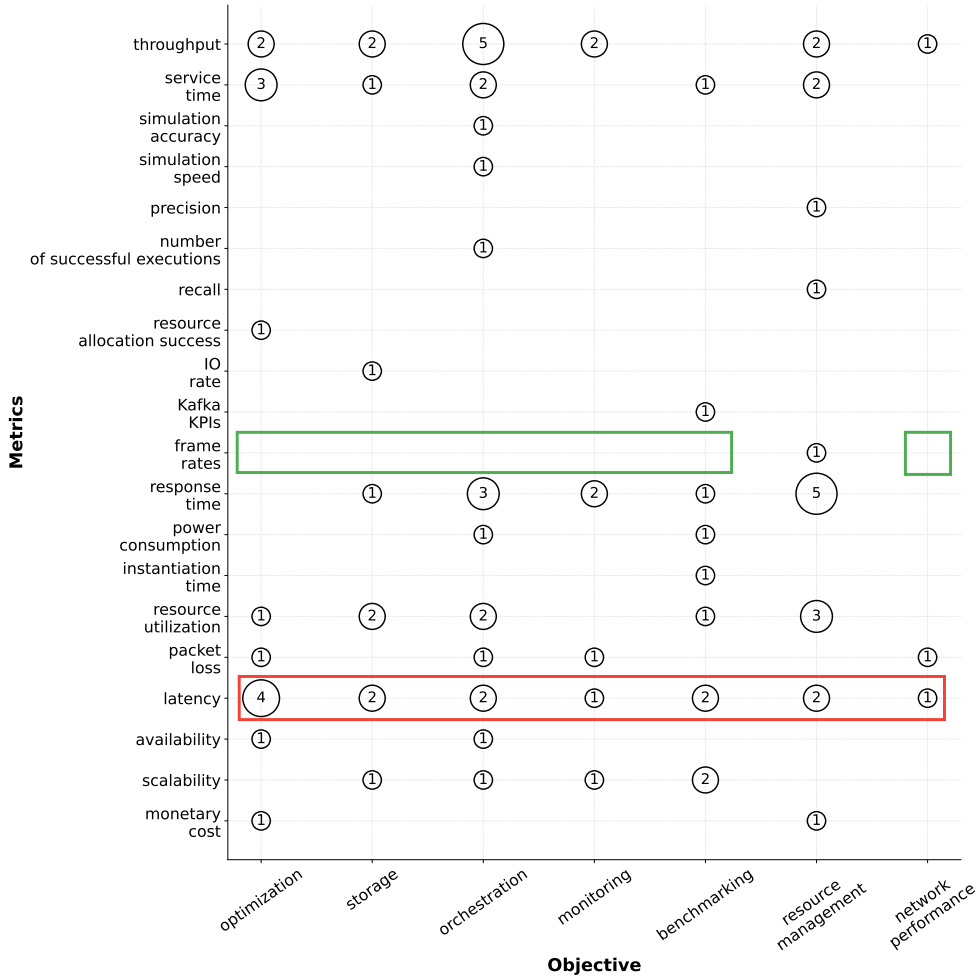


Fig. 10: Combination between metrics and objectives.

edge, as highlighted in [PS14]. Latency, a key performance indicator, encompasses communication latency among microservices and service latency, which includes inter-microservice latency and processing time, impacting overall functionality [PS11]. In the realm of telecommunication networks, latency, packet loss, and throughput are fundamental metrics for performance evaluation, with delay representing the time taken for packet transmission across a network path [PS02]. Furthermore, in latency-sensitive applications, execution time and invocation delay influence the software, hardware capabilities, and network-specific parameters, highlighting the importance of measuring and optimizing latency for efficient operations [PS04].

Potential Gaps

There exists a notable gap in research concerning the utilization of frame rate as a metric. While metrics like latency and throughput have garnered attention, frame rate, which is particularly relevant in applications involving multimedia content or real-time graphics, remains largely unexplored. This gap represents an opportunity for researchers to explore the impact of frame rate on user experience, system performance, and resource utilization within cloud-native environments. By incorporating frame rate analysis into evaluation methodologies, researchers can gain valuable insights into optimizing cloud-native applications for tasks such as video streaming, gaming, video calling, or virtual reality. Addressing this research gap holds promise for enhancing the responsiveness, visual quality, and overall effectiveness of cloud-native applications across a diverse range of use cases.

5.5 Software x Objective

Figure 11 presents the relationship between software and objectives.

Prevalent Research Domains

Kubernetes remains a focal point, particularly in the domain of resource management. Researchers are exploring mechanisms of orchestrating and optimizing resource allocation within Kubernetes clusters. From fine-tuning scheduling algorithms to exploring ways of scaling resources dynamically, the pursuit is to enhance efficiency, resilience, and cost-effectiveness in containerized environments. This focus reflects Kubernetes' growing importance as the backbone of modern cloud-native architectures, driving continuous to meet the evolving demands of scalable and reliable applications. The platform orchestrates applications, enforces resource limits, and optimizes resource consumption through features like the Completely Fair Scheduler bandwidth control [PS10]. Research projects explore the performance of cloud-native frameworks, comparing platforms like Docker and Kubernetes in terms of completion time and resource utilization [PS20]. Additionally, Kubernetes offers various resources, such as Services, Ingresses, and ConfigMaps, for efficient deployment and management of components in different environments [PS09]. The use of Kubernetes as a container orchestrator simplifies the deployment and management of distributed systems like Elasticsearch, enhancing scalability and reliability in complex architectures [PS21]. Furthermore, Kubernetes is essential for managing containerized systems in cloud environments, ensuring efficient resource assignment and lifecycle management for application containers running in pods on cluster nodes [PS19].

Potential Gaps

Despite the extensive utilization of TensorFlow, PyTorch, Spark, and Grafana in resource management objectives within cloud-native ecosystems, a conspicuous gap emerges in their application towards performance evaluation of cloud-native applications. While these tools excel in facilitating efficient resource allocation and monitoring within Kubernetes clusters, their potential for evaluating the performance of cloud-native applications still needs to be explored. Incorporating these frameworks into the

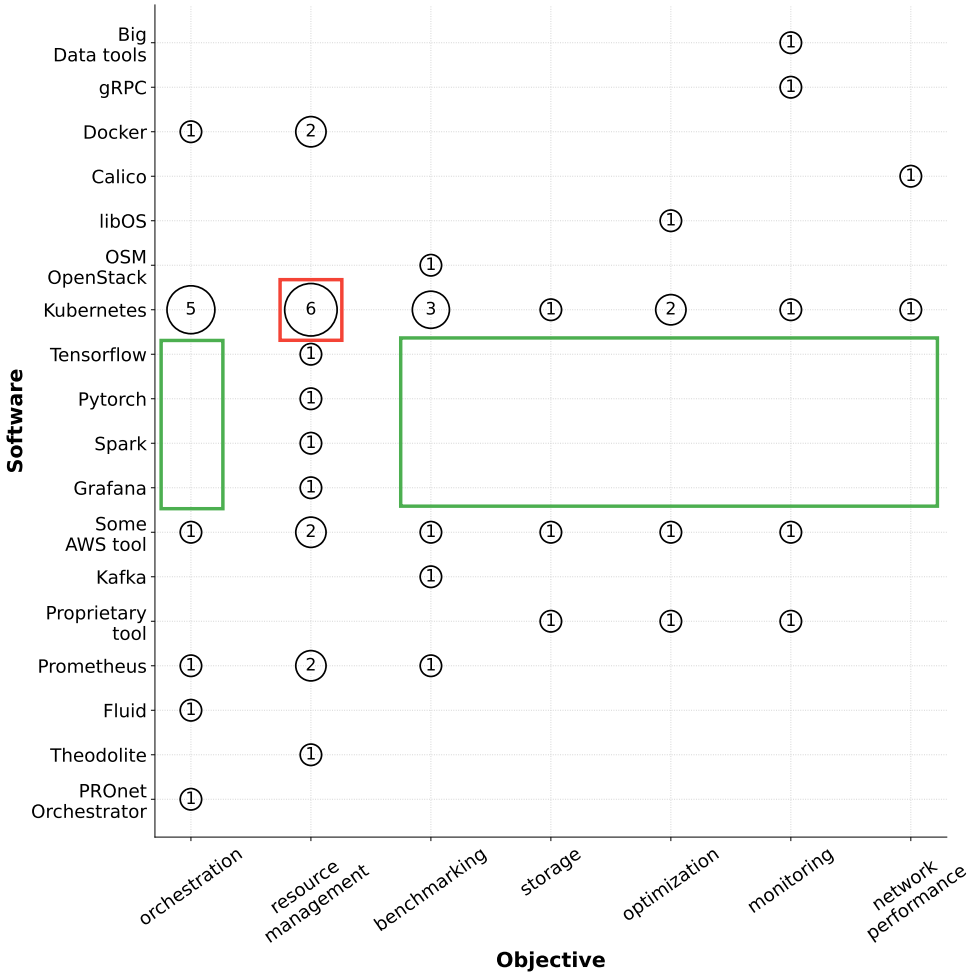


Fig. 11: Combination between software and objective.

evaluation process could offer invaluable insights into the runtime behavior, scalability, and optimization opportunities of cloud-native applications, enabling developers to make informed decisions to enhance their overall performance. Grafana, for example, is a powerful tool that finds motivation for its usage in various contexts. Grafana is utilized for monitoring and visualizing data in real time, making it valuable for analyzing complex systems and identifying trends efficiently.

5.6 Metric x Software

Figure 12 presents the relationship between metric and software.

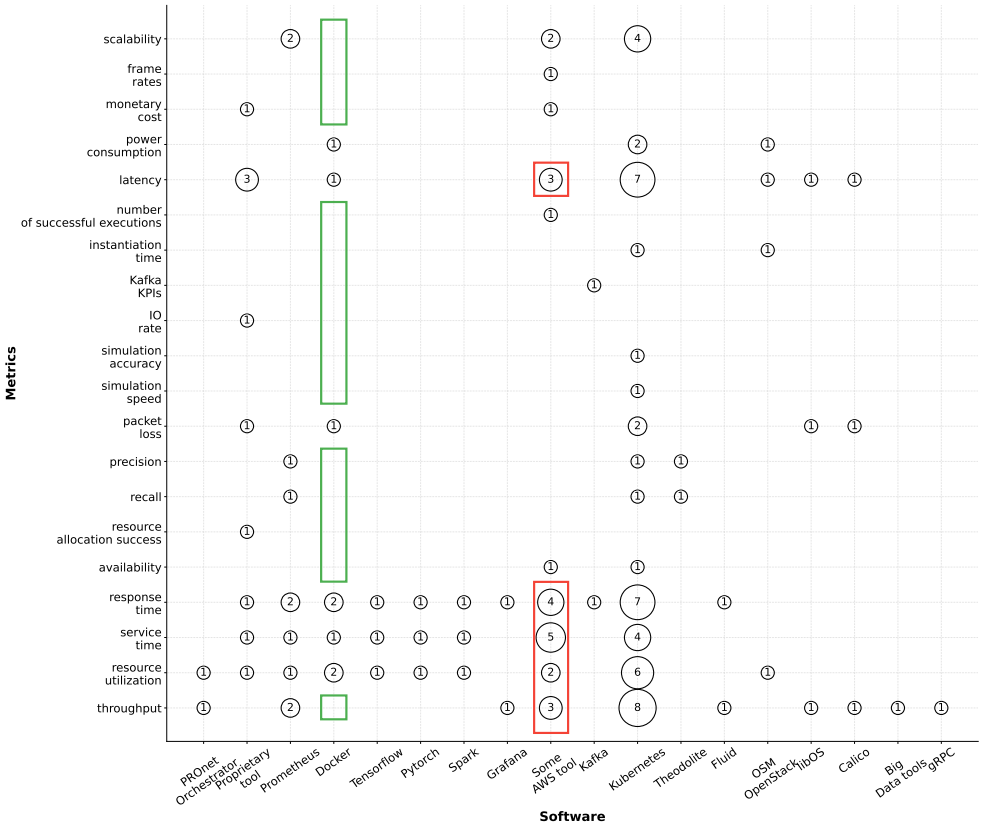


Fig. 12: Combination between metrics and software.

Prevalent Research Domains

The focus once again resides on AWS tools, with a particular emphasis on service time. Scholars and practitioners are deeply engaged in exploring the nuances of service time metrics within the AWS ecosystem, aiming to refine and optimize the delivery of cloud services. From analyzing the latency of various AWS services to investigating strategies for minimizing service time fluctuations, the pursuit is directed toward enhancing the reliability and responsiveness of cloud-based applications. This renewed emphasis underscores the critical role that service time plays in shaping the user experience and operational efficiency within cloud environments, propelling ongoing efforts to innovate and refine AWS tools for optimal performance and service delivery.

Potential Gaps

Despite the adoption of Docker in cloud-native environments, there exists a noticeable gap in its utilization for the performance evaluation of cloud-native applications. While Docker excels in containerization and deployment, its potential for performance assessment still needs to be explored. Current evaluation practices often

focus on resource utilization metrics and container orchestration dynamics, overlooking deeper insights into application-level performance characteristics. Incorporating Docker into performance evaluation frameworks could offer valuable insights into runtime behavior, scalability, and optimization opportunities specific to containerized applications. Closing this gap presents an exciting opportunity for researchers and practitioners to enhance the effectiveness of performance evaluation methodologies in cloud-native environments, ultimately leading to more efficient application deployments. For example, a potential metric that needs to be explored is throughput.

Docker is motivated by its high performance compared to competitors such as VMs and Unikernels, especially in reducing startup and serving time [PS07]. It is designed to deploy and run distributed applications efficiently without launching entire virtual machines. It is ideal for cloud-native applications and microservices managed on elastic infrastructure through agile processes [PS20]. Docker's containerization technology manages AI model training platforms with GPU hardware in cloud environments, optimizing resource utilization for data-hungry and computation-intensive workloads such as deep learning [PS06]. Furthermore, Docker's open-source containerization service, coupled with orchestration systems such as Kubernetes, allows efficient resource utilization in cloud environments, making it essential for implementing diverse services through containerized systems in the cloud [PS26].

6 Research Challenges and Directions

The performance evaluation of cloud-native applications presents several challenges that researchers and practitioners need to address. Some of the main challenges include:

Dynamic Nature

Cloud-native applications, such as those leveraging technologies like Docker and Kubernetes, are dynamic. Components like containers and microservices can scale up or down based on demand, leading to a constantly changing application environment. This dynamic nature poses a challenge for performance evaluation, as traditional metrics may not accurately reflect real-time performance under varying loads. Cloud-native applications, leveraging technologies like Docker and Kubernetes, exhibit a dynamic nature, with components such as containers and microservices continuously scaling based on demand [PS20] [PS10]. This dynamism poses challenges for performance evaluation, as traditional metrics may need to capture real-time performance fluctuations under varying workloads effectively. Cloud-native platforms like Kubernetes allow for dynamic resource allocation, but issues like CPU throttling can impact overall system performance [PS05]. Additionally, in cloud-native network functions, the kernel space is identified as having more potential for network acceleration, especially in virtual machine deployments [PS23]. Understanding and adapting to this dynamic environment helps optimize the performance of cloud-native applications in response to changing demands and resource constraints.

Scalability

Ensuring consistent performance as applications scale horizontally (adding more instances) or vertically (increasing resources per instance) is not just a nice to have but a necessity. Performance benchmarks need to be adaptable and account for these scaling scenarios to provide meaningful insights. Scalability is a fundamental requirement for cloud-native applications, which are specifically engineered to operate across dynamic and scalable cloud infrastructures [PS20]. Cloud-native tools such as Kubernetes play an essential role in enabling scalability by efficiently orchestrating application containers [PS18]. Leveraging mechanisms like Objects Lambda and S3 Select can improve the performance of cloud-native distributed object storage. These mechanisms optimize data processing and movement closer to storage nodes, enhancing compute efficiency and performance gains [PS19]. Additionally, the scalability of cloud-native network functions faces challenges in network performance, particularly in packet processing delays. Solutions focus on accelerating processing efficiency in both the user and kernel spaces [PS12]. Overall, the scalability of cloud-native applications is a critical consideration in ensuring their ability to handle increasing workloads and demands in dynamic cloud environments.

Complexity of Ecosystem

Cloud-native applications often leverage a variety of tools and technologies, such as container orchestration platforms (e.g., Kubernetes), service meshes, and serverless computing frameworks. Evaluating performance across this diverse ecosystem requires understanding and integrating different metrics and tools. The evaluation of performance in cloud-native applications is linked to the complexity of the ecosystem surrounding them. As highlighted in [PS30], navigating the intricate ecosystem surrounding cloud-native applications presents a challenging yet essential consideration. Factors such as the network infrastructure, deployment environments, integration with external services, and the overall architecture can impact the scalability, reliability, and resource utilization of cloud-native applications. Understanding and assessing this complexity is not only essential but also an exciting part of conducting accurate performance evaluations, identifying bottlenecks, optimizing resource allocation, and ensuring the efficient operation of cloud-native applications in diverse and dynamic environments.

Monitoring and Observability

Because cloud-native applications are distributed, it becomes critical to monitor performance metrics across multiple components and services. Ensuring observability without impacting application performance itself is a challenge. The impact of monitoring and observability on the scalability of cloud-native applications is significant. In [PS19], it is highlighted that observability is vital for tracing application execution and detecting malfunctions and anomalies in cloud-based applications and IoT use cases. Observability refers to the ability to understand a system's internal state based on its external outputs. In the context of cloud-native applications, this means being able to trace the execution of various components and services, detect

any malfunctions or anomalies, and identify the root causes of errors. Efficient monitoring and observability enable the identification of performance issues, bottlenecks, and resource demands, allowing for proactive scaling and optimization to meet the increasing workload demands. Monitoring and observability, by collecting and analyzing relevant data, contribute to enhancing the performance, resource utilization, and overall scalability of cloud-native applications.

Multi-Tenancy Issues

In multi-tenant environments, where multiple users or applications share the same infrastructure resources, ensuring fair allocation of resources and evaluating performance in isolation from other tenants is critical. In the context of cloud-native applications, multi-tenancy refers to a cloud system's capability to support multiple users or tenants sharing the same resources and infrastructure. In a multi-tenant environment, ensuring equitable resource allocation, preventing performance bottlenecks, and maintaining data isolation are essential factors that influence the overall performance of cloud-native applications. Issues such as noisy neighbors, where one tenant's high resource demands affect other tenants' performance, can arise and lead to performance fluctuations. Moreover, security concerns related to data privacy and integrity increase in multi-tenant environments, necessitating measures to ensure secure isolation and compliance with data regulations. Addressing multi-tenancy issues requires careful monitoring, workload management, and performance optimization strategies tailored to the unique requirements of cloud-native applications in shared environments [PS06].

Security and Compliance

Performance evaluation must also consider security aspects such as data protection, access control, and compliance requirements (e.g., General Data Protection Regulation). Ensuring that performance benchmarks do not compromise security or regulatory compliance adds another layer of complexity. Security plays a critical role in the performance evaluation of cloud-native applications [PS03]. Security is not just a side note in the performance evaluation of cloud-native applications. As these applications are deployed in dynamic and shared environments, assessing their performance involves considerations beyond speed and efficiency. Security concerns such as data protection, access control, and threat detection directly impact the application's overall performance and reliability. Vulnerabilities in security measures can lead to breaches, data loss, or service interruptions, ultimately affecting the application's performance metrics and user experience. By integrating security protocols and conducting thorough security evaluations during performance assessments, organizations can ensure the safety and integrity of their cloud-native applications while optimizing their performance for enhanced operational efficiency and user trust.

Resource Utilization Optimization

Optimizing resource utilization while maintaining performance levels is essential for cost-effectiveness and efficient operation of cloud-native applications. Performance evaluations need to balance resource efficiency with performance metrics. Efficient

resource utilization is vital in evaluating the performance of cloud-native applications [PS29]. Efficient utilization of resources such as CPU, memory, and storage directly impacts the overall performance, scalability, and cost-effectiveness of applications running in cloud environments. Optimizing resource utilization ensures that the allocated resources are utilized effectively, minimizing wastage and enhancing the application's responsiveness and stability. By fine-tuning resource allocation based on workload demands, organizations can improve the application's efficiency, reduce latency, and enhance the user experience. Additionally, optimizing resource utilization contributes to better resource management, increased system reliability, and optimized cost efficiency in cloud-native environments. Overall, prioritizing resource utilization optimization is essential for maximizing the performance and reliability of cloud-native applications while effectively managing operational costs.

Heterogeneity of Workloads

Cloud-native applications may handle a wide range of workloads, from batch processing to real-time data streaming. Evaluating performance across these heterogeneous workloads requires adaptable performance evaluation strategies. The diversity of workloads impacts the performance evaluation of cloud-native applications [PS25]. By encompassing a diverse range of workloads characterized by varying attributes, such as data intensity, metadata intensity, read/write intensity, and specific application requirements. Organizations can gain understanding of their application's scalability, efficiency, and responsiveness under different operational scenarios. Evaluating cloud-native applications with heterogeneous workloads allows for a more holistic assessment of performance capabilities, enabling organizations to tailor their resource allocations, optimize system configurations, and enhance overall application robustness. This approach ensures that cloud-native applications are tested under realistic conditions, reflecting the dynamic nature of modern workloads in cloud environments. We did not explore this aspect in the mapping because the papers utilized a wide variety of workloads, which made categorization and analysis challenging. Therefore, this aspect could be explored further in future research endeavors.

7 Conclusion

This paper has provided a detailed exploration of key themes and insights derived from the individual analysis of research questions related to metrics, software programs, objectives, and primary research contributions in cloud-native performance evaluation. Through meticulous examination, we have highlighted trends and preferences within the literature. Metrics such as latency, response time, and throughput emerged as critical indicators of system performance, offering insights into network efficiency and user experience. Software programs like Kubernetes and AWS tools manages and optimizes cloud-native applications, underscoring their adoption and impact. Objectives such as resource management and orchestration were identified as primary focuses in enhancing scalability, reliability, and cost-effectiveness across

cloud environments. Moreover, the emphasis on performance evaluation as a contribution underscores its importance in advancing methodologies and tools for assessing cloud-native application performance.

The paper also presented a multidimensional analysis across various criteria. By exploring intersections of metrics, objectives, and software tools, this study has identified both prevalent research domains and potential gaps for future exploration. The prevalence of throughput metrics alongside methodologies underscores their critical role in enhancing system performance and scalability. Conversely, gaps in research, such as the under-exploration of metrics like packet loss, availability, and monetary cost, highlight opportunities for developing novel evaluation methods. Gaps remain, particularly in leveraging tools like Prometheus for performance assessment and exploring Docker's potential in application-level performance evaluations.

This study identifies three central challenges to its validity: (i) the possibility that the research questions may not cover all relevant studies on cloud-native application performance, (ii) the risk of selection bias that could lead to the exclusion of pertinent research, and (iii) the subjective nature involved in extracting data. To mitigate these challenges, the study employed a research strategy and involved collaborative classification by multiple co-authors, and the paper project benefitted from external expert supervision. These efforts aimed to ensure a thorough exploration of the literature and enhance the reliability of the study's findings.

The exploration of cloud-native application performance evaluation presents several compelling avenues for future research. Future research may dive into the underexplored challenges related to cloud-native performance evaluation. The new research can focus on the research challenges presented in the last section. An essential focus is developing adaptive metrics and evaluation methodologies that accurately reflect the dynamic nature of cloud-native environments. Another promising direction is advancing monitoring and observability techniques tailored for cloud-native applications.

8 Declarations

Ethical Approval

Not applicable

Competing interests

The authors have no relevant financial or non-financial interests to disclose.

Authors' contributions

The authors have contributed equally to this work.

Funding

Not applicable

Availability of data and materials

Data sharing not applicable.

References

[1] Cloud native applications market. https://www.marketsandmarkets.com/Market-Reports/cloud-native-applications-market-53431081.html. Accessed: June 2024.

[2] Ahmadi, S. 2023. Elastic data warehousing: Adapting to fluctuating workloads with cloud-native technologies. *Journal of Knowledge Learning and Science Technology ISSN: 2959-6386 (Online) 2*(3): 282–301 .

[3] Alonso, J., L. Orue-Echevarria, V. Casola, A.I. Torre, M. Huarte, E. Osaba, and J.L. Lobo. 2023. Understanding the challenges and novel architectural models of multi-cloud native applications–a systematic literature review. *Journal of Cloud Computing 12*(1): 6 .

[4] Csáti, G., I. Pelle, and L. Toka 2022. Optimizing performance and resource consumption of cloud-native logging application stacks. In *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–4. IEEE.

[5] Deng, S., H. Zhao, B. Huang, C. Zhang, F. Chen, Y. Deng, J. Yin, S. Dustdar, and A.Y. Zomaya. 2024. Cloud-native computing: A survey from the perspective of services. *Proceedings of the IEEE* .

[6] Huang, S.Y., C.Y. Chen, J.Y. Chen, and H.C. Chao. 2023. A survey on resource management for cloud native mobile computing: opportunities and challenges. *Symmetry 15*(2): 538 .

[7] Kanchepu, N. 2023. Cloud-native architectures: Design principles and best practices for scalable applications. *International Journal of Sustainable Development Through AI, ML and IoT 2*(2): 1–21 .

[8] Kitchenham, B., S. Charters, et al. 2007. Guidelines for performing systematic literature reviews in software engineering.

[9] Kosińska, J., B. Baliś, M. Konieczny, M. Malawski, and S. Zieliński. 2023. Toward the observability of cloud-native applications: The overview of the state-of-the-art. *IEEE Access* 11: 73036–73052 .

[10] Kratzke, N. and P.C. Quint. 2017. Understanding cloud-native applications after 10 years of cloud computing-a systematic mapping study. *Journal of Systems and Software* 126: 1–16 .

[11] Matthew, O.O., D.O. Olabanji, and T. Fitch. 2023. Multi-tenancy in cloud-native architecture: A systematic mapping study. *WSEAS Transactions on Computers 22*(4): 25–43 .

[12] Moravcik, M., M. Kontsek, P. Segec, and D. Cymbalak 2022. Kubernetes-evolution of virtualization. In *2022 20th International Conference on Emerging*

eLearning Technologies and Applications (ICETA), pp. 454–459. IEEE.

[13] Odun-Ayo, I., R. Goddy-Worlu, L. Ajayi, B. Edosomwan, and F. Okezie 2019. A systematic mapping study of cloud-native application design and engineering. In *Journal of Physics: Conference Series*, Volume 1378, pp. 032092. IOP Publishing.

[14] Paredes, J.R.B. and F. López-Pires 2022. A survey on billing models for cloud-native applications. In *Conference on Cloud Computing, Big Data & Emerging Topics*, pp. 20–30. Springer.

[15] Petersen, K., S. Vakkalanka, and L. Kuzniarz. 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and software technology* 64: 1–18 .

[16] Rahaman, M.S., A. Islam, T. Cerny, and S. Hutton. 2023. Static-analysis-based solutions to security challenges in cloud-native systems: Systematic mapping study. *Sensors 23*(4): 1755 .

[17] Rahaman, M.S., S.N. Tisha, E. Song, and T. Cerny. 2023. Access control design practice and solutions in cloud-native architecture: A systematic mapping study. *Sensors 23*(7): 3413 .

[18] Shah, S.D.A., M.A. Gregory, and S. Li. 2021. Cloud-native network slicing using software defined networking based multi-access edge computing: A survey. *IEEE Access* 9: 10903–10924 .

[19] Skarin, P., W. Tärneberg, K.E. Årzén, and M. Kihl 2020. Control-over-the-cloud: A performance study for cloud-native, critical control systems. In *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, pp. 57–66. IEEE.

[20] Theodoropoulos, T., L. Rosa, C. Benzaid, P. Gray, E. Marin, A. Makris, L. Cordeiro, F. Diego, P. Sorokin, M.D. Girolamo, et al. 2023. Security in cloud-native services: A survey. *Journal of Cybersecurity and Privacy 3*(4): 758–793 .

[21] Vaño, R., I. Lacalle, P. Sowiński, R. S-Julián, and C.E. Palau. 2023. Cloud-native workload orchestration at the edge: A deployment review and future directions. *Sensors 23*(4): 2215 .

[22] Zhang, J., W. Jiang, B. Tang, H. Ma, L. Cao, Z. Jiang, Y. Nie, F. Wang, L. Zhang, and Y. Liang. 2023. Cdsben: Benchmarking the performance of storage services in cloud-native database system at bytedance. *Proceedings of the VLDB Endowment 16*(12): 3584–3596 .

Artigos publicados em 2026

IACR Communications in Cryptology
ISSN 3006-5496, Vol. 2, No. 4, 9 pages.

https://doi.org/10.62056/akmpdk5vt

Check for updates

Matrix Polynomial Attack on the Megrelishvili Key Exchange Protocol

Raul Caram de Assis and Thiago do Rêgo Sousa

CEPESC, Brasília, Brazil

Abstract. In this article we explain what the Megrelishvili Key Exchange Protocol (MKE) is and compare it briefly to the older Diffie-Hellman Key Exchange Protocol (DHKE). We clarify how the security of the MKE relates to the hardness of the Megrelishvili Vector-Matrix Problem (MVMP) and also to a generalization of this problem, introduced by us, which we call the Matrix Polynomial Problem (MPP). There were previous sub-exponential time attacks on the MKE [AW16, AMP18], the first applying to arbitrary instances and the latter requiring the public matrix M to be diagonalizable — as well as an attack on the one-way function used in the MKE [Meg14] that required special conditions, but when applicable would allow to break the protocol in polynomial time according to our running-time analysis. Here we adopt a bottom-up approach to construct for the first time, as far as our review of the relevant literature reveals, a polynomial time algorithm that solves any instance of the MKE through its reduction to a MPP. We also introduce some key concepts from linear algebra and Krylov subspace theory into the context of the MKE, allowing for a clearer understanding of our attack.

Keywords: Megrelishvili Protocol · Key Exchange Protocol · Diffie-Hellman · Group Action · Minimal Polynomial · Krylov Subspaces

1 Introduction

Inspired by the famous DHKE [DH76], many other protocols have been suggested for the purpose that two parties, without any previous shared knowledge, could define for themselves, and only themselves, a secret key and without ever explicitly communicating that secret or any information that could efficiently be used to deduce it. Those parties are commonly named 'Alice' and 'Bob', assumed to be communicating through an insecure channel that is being wiretapped by a third person 'Eve'. Alice and Bob share between themselves some public information that is then used by both to derive a secret sequence that becomes known to Alice and Bob but not Eve. Of course if the secret sequence can be easily calculated from the public information the protocol's purpose is defeated.

Megrelishvili, Chelidze and Chelidze proposed in 2006 [MCC06] a variant of the DHKE that we refer to as the MKE [AW16, Arz16, AMP18, Arz18]. It is a matrix-based variant built upon taking a matrix $M \in GL(n, p)$, the group of invertible $n \times n$ matrices over $\mathbb{Z}_p$, a non-zero $1 \times n$ vector $\mathbf{v}$ over $\mathbb{Z}_p$ and then have the subgroup generated by M, $\langle M \rangle$, to act on the set of non-zero row vectors $(\mathbb{Z}_p^n)^* = \mathbb{Z}_p^n \setminus \mathbf{0}$. It relies on vector-matrix products rather than exponentiation of group elements, like the original DHKE. A detailed description of the MKE is found in Section 1.4.

There are also previous works that have either used the MKE as a key exchange protocol or have derived other protocols from it. More specifically, the protocol is generalized for

E-mail: raulcaram21@gmail.com (Raul Caram de Assis), thiagodoregosousa@gmail.com (Thiago do Rêgo Sousa)

This work is licensed under a "CC BY 4.0" license.
Received: 2025-10-07 Accepted: 2025-12-02

multi-party computation [AW17], group key distribution [Arz17] and for one-time and a few-time digital signatures [Arz19]. The works, [IWI+21, WJHK21], each designed a protocol that uses the MKE to establish a secret shared key to be used in a symmetric algorithm for encrypting general purpose or medical images.

1.1 Related works

Previously [Meg14] dealt with the one-way function $f_{\mathbf{v},M}(b) = \mathbf{v}M^b = \mathbf{w}_B$ (a central cryptographic primitive of the MKE), where $\mathbf{v}$ and M have entries over the field $\mathbb{Z}_p$ and M is an invertible matrix of maximum order. It states that $f_{\mathbf{v},M}$ is broken as a consequence of $\{M^0, M^1, \ldots, M^{n-1}\}$ spaning $\langle M \rangle$ and provides a toy example in $\mathbb{Z}_2$ where the set $S = \{\mathbf{v}, \mathbf{v}M^1, \ldots, \mathbf{v}M^{n-1}\}$ is linearly independent and can be used, along with $\mathbf{w}_B$, to construct a linear system with a unique solution, i.e. a unique linear combination of the set S that results in $\mathbf{w}_B$. This solution is identified and then used to recover M^b. Naturally this compromise the MKE as M^b can then be used to recover the shared secret key $\mathbf{w}_K$.

Other former works of cryptanalysis of the MKE centered on analyzing the hardness of the Megrelishvili Vector-Matrix Problem (MVMP, described in subsection 1.5) [Arz16, AW16, AMP18], which is the analogue of the discrete logarithm problem (DLP) to the MKE.

The article [Arz16] introduces the definition of the MVMP and delineates its relation to the security of the MKE. It provides a complexity analysis of solving the MVMP by exhaustive search showing it takes time exponential in n. On the other hand, the article [AW16] constructs two collision algorithms to solve the MVMP, the first is deterministic and an adaptation of the baby-step giant-step to the context of the MKE, the second is similar to the first but randomized. They both have an expected running-time complexity that is at least exponential in $n/2$.

Lastly, in case the public matrix M is diagonalizable, the article [AMP18] constructs an attack on the MKE by doing a straightforward adaptation of the ideas present in [MW97], the latter shows a probabilistic polynomial-time reduction of the DLP version in $GL(n, q)$ to the DLP version in small extensions of finite fields, while the former provides a polynomial-time reduction of the MVMP to a set of at most n finite field DLPs. This attack would still take sub-exponential time in n using a classical computer but could be solved in polynomial time by a quantum computer.

1.2 Our contributions

- §2.1 — We show that breaking any instance of the MKE can be reduced to a more general mathematical problem than the MVMP, which we call the MPP.

- §2.3 — We construct a classical (i.e. non-quantum) algorithm that solves the MPP in polynomial time and is able to deduce from the public parameters **pub** the shared secret key $\mathbf{w}_K$. This solves one of the questions raised in [AW16, Section 6], showing that the problem of finding $\mathbf{w}_K$ key from **pub** in the MKE admits, in every case, an efficient algorithm that does not require solving the MVMP;

- §2.3 — We do a complexity analysis of our key-recovery algorithm to justify the claim for its polynomial running-time;

- §2.3 - We show how the approach of [Meg14] can be viewed as a special case of our algorithm, thereby clarifying its boundaries and situating it within the general attack framework presented here. His attack, as delineated, expects to recover M^b and requires the constructed linear system to be consistent and independent. We formalize and present an attack to the MKE that extends the ideas in [Meg14], allowing the

recovery of $\mathbf{w}_K$ for any MKE instance and without requiring (in the general case) neither the knowledge of M^b nor his original linear system to be independent.

- §2.3 — We provide theoretical justifications for the space of solutions to the MPP being an affine subspace and how it connects to theory of Krylov subspaces.

1.3 Structure of the article

In Section 1 we present the necessary background for understanding the object of our article, the MKE, as well as a summary of what has been developed about the protocol from its inception in 2006 until the present day. Section 2 is reserved to our original research work, except for the preliminaries in subsection 2.2. Then we end with Section 3 for our conclusions.

1.4 The Megrelishvili Protocol

The MKE consists of a series of steps that can be briefly summarized as follows. All communication between Alice and Bob is assumed to be closely monitored by Eve:

- First, Alice and Bob talk to each other and reach an agreement for the public parameters $n, p \in \mathbb{N}$, with p prime so as to make $\mathbb{Z}_p$ a field, as well as $M \in GL(n, p)$ and $\mathbf{v} \in (\mathbb{Z}_p^n)^*$. Possibly, in some variants the public parameters may de defined by a Trusted Third Party to avoid a man-in-the-middle attack, either way we assume Eve has access to these parameters [Arz18];
- Second, Alice generates her private key $a \in \mathbb{Z}$ and calculates

$$\mathbf{w}_A = \mathbf{v}M^a \tag{1}$$

  and, independently from her, Bob generates his own private key $b \in \mathbb{Z}$ and calculates

$$\mathbf{w}_B = \mathbf{v}M^b; \tag{2}$$

- Third, Alice sends $\mathbf{w}_A$ to Bob and receives $\mathbf{w}_B$ from Bob;
- Lastly, both Alice and Bob compute their shared secrets $\mathbf{w}_{K_A}$ and $\mathbf{w}_{K_B}$, respectively, as:

$$\mathbf{w}_{K_A} = \mathbf{w}_B M^a = (\mathbf{v}M^b)M^a = \mathbf{v}M^{a+b}.$$
$$\mathbf{w}_{K_B} = \mathbf{w}_A M^b = (\mathbf{v}M^a)M^b = \mathbf{v}M^{a+b}.$$

  Their shared key is then established as

$$\mathbf{w}_K = \mathbf{w}_{K_A} = \mathbf{w}_{K_B}. \tag{3}$$

The above protocol is broken if Eve is able to know $\mathbf{w}_K$ from the public parameters

$$\mathbf{pub} = (n, p, M, \mathbf{v}, \mathbf{w}_A, \mathbf{w}_B), \tag{4}$$

that is, from the data transmitted through the channel. We stress every operation related to the MKE through this article is done over $\mathbb{Z}_p$, so every *resulting value* of addition and multiplication of matrices is to be understood as reduced modulo p. However the exponents over the matrices, such as e in M^e, are in $\mathbb{Z}$ not in $\mathbb{Z}_p$ as they refer to an operation (multiplication by M or M^{-1}) being done a certain amount of times.

If we take any matrix $M \in GL(n, p)$, the powers of M form a finite cyclic subgroup $\langle M \rangle$ of $GL(n, p)$. We use the expression $\mathrm{ord}(M)$ to denote the cardinality of $\langle M \rangle$. Thus any exponent, including negative ones, can equivalently be understood as reduced modulo $\mathrm{ord}(M)$ and thus as a number in $\{0, 1, \ldots, \mathrm{ord}(M) - 1\}$.

1.5 The Megrelishvili Vector-Matrix Problem

If Eve knows **pub** and is able to find either e_a or $e_b \in \mathbb{Z}$ such that $\mathbf{w}_A = \mathbf{v} \cdot M^{e_a}$ or $\mathbf{w}_B = \mathbf{v} \cdot M^{e_b}$ then she can easily break the protocol and compute the shared key defined in (3) by doing:

- In the first case, $\mathbf{w}_B \cdot M^{e_a} = \mathbf{v} \cdot M^b \cdot M^{e_a} = \mathbf{v} \cdot M^{e_a} \cdot M^b = \mathbf{w}_A \cdot M^b = \mathbf{w}_K$, or;
- in the second case, $\mathbf{w}_A \cdot M^{e_b} = \mathbf{v} \cdot M^a \cdot M^{e_b} = \mathbf{v} \cdot M^{e_b} \cdot M^a = \mathbf{w}_B \cdot M^a = \mathbf{w}_K$.

The MVMP, introduced by [Arz16], is defined as the general problem of: Given a matrix $M \in GL(n, p)$ and vectors $\mathbf{v}$, $\mathbf{w} = \mathbf{v}M^j \in (\mathbb{Z}_p^n)^*$ for some $j \in \mathbb{Z}$, find an exponent $e \in \mathbb{Z}$ such that $\mathbf{w} = \mathbf{v} \cdot M^e$.

Clearly the security of the MKE depends at least on the hardness of the MVMP, as any solution to the MVMP allows the recovery of $\mathbf{w}_K$.

1.6 Shor's Algorithm and Group Action

In the DHKE, Alice and Bob agree on a finite field $\mathbb{Z}_p$ and $g \in \mathbb{Z}_p$. For cryptographic security, we take g to be a generator for $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus 0$ meaning the cyclic group $\langle g \rangle$ has $p-1$ distinct elements and coincides with $\mathbb{Z}_p^*$. Alice chooses a secret exponent $a \in \mathbb{Z}$ and sends $g^a \pmod p$ to Bob, similarly Bob chooses a secret exponent $b \in \mathbb{Z}$ and sends $g^b \pmod p$ to Alice. From this point both parties can compute $g^{ab} = (g^a)^b = (g^b)^a$ which is their shared key.

The security of the DHKE depends on the difficulty of solving the DLP in the multiplicative group $\mathbb{Z}_p^*$, which is the problem of given g^a, for a known $g \in \mathbb{Z}_p^*$ and unknown $a \in \mathbb{Z}$, deduce the value of $a \mod (p-1)$.

If Eve observes a value $g^a \in \langle g \rangle$ she knows how to calculate any power of g^a, that is, she can find elements in the multiplicative cyclic group $\langle g \rangle = \{g^0, g^1, \ldots, g^{p-2}\}$ of the form $(g^a)^x = g^{ax}$. Note g^a is in position $(a \mod (p-1)) + 1$ in the tuple $(g^0, \ldots, g^{p-2})$, Eve can calculate g^{ax} and know the element in position $(ax \mod (p-1)) + 1$ of that tuple without knowing a.

In contrast the MKE makes use of the group action $\xi : \langle M \rangle \times (\mathbb{Z}_p^n)^* \to (\mathbb{Z}_p^n)^*$ that takes, for any $e \in \mathbb{Z}$ and $\mathbf{v} \in (\mathbb{Z}_p^n)^*$, $(M^e, \mathbf{v})$ to $\mathbf{v}M^e$. Shor's algorithm [Sho94] version for the DLP runs in quantum polynomial time assuming, among other things, it can exponentiate or multiply elements of the group efficiently [Chi17, Ch. 5]. These types of constructions give interesting candidate problems for post-quantum cryptographic systems because the transmitted values between Alice and Bob are not in a multiplicative group. If we consider the tuple $(\mathbf{v}M^0, \ldots, \mathbf{v}M^{\mathrm{ord}(M)-1})$, given an exponent x and $\mathbf{v}M^a$, for any a, it is not as straightforward to find the element $\mathbf{v}M^{ax}$ in position $(ax \mod \mathrm{ord}(M)) + 1$ without finding a first. So it seems to create a sort of chicken-egg problem where in order to run Shor's algorithm efficiently to find the value of a, you must first be able to "exponentiate" $\mathbf{v}M^a$, but to do it you must, at first sight, know the value of a.

2 Breaking the MKE

2.1 The Matrix Polynomial Problem

Let $\mathcal{V}$ be the linear space formed by matrices that are polynomial in M and recall the definitions of $\mathbf{w}_A$, $\mathbf{w}_B$ and **pub** from (1), (2) and (4), respectively. We will define and solve the following matrix polynomial problem.

$$\text{MPP : Given } \mathbf{pub} \text{ find } Q \in \mathcal{V} \text{ such that } \mathbf{v}Q = \mathbf{w}_B.$$

We use $\mathbf{w}_B$ above in the problem definition rather than $\mathbf{w}_A$ for no special reason. Any solution to the MPP allows for efficiently recovering $\mathbf{w}_K$. To prove our assertion, assume Eve finds a matrix Q satisfying the conditions above. Since every element in $\mathcal{V}$ commutes with M, she can calculate $\mathbf{w}_A Q = \mathbf{v}M^a Q = \mathbf{v}QM^a = \mathbf{w}_B M^a = \mathbf{w}_K$ and retrieve the shared secret. In fact, these restrictions on the matrix Q could be relaxed to only require that $QM = MQ$ and $\mathbf{v}Q = \mathbf{w}_B$.

Note that, by solving the MVMP, Eve would know b (or a) and thus she is able to calculate M^b (or M^a) which are particular solutions to the MPP (or its analogue using $\mathbf{w}_A$). Our observation shows that not only do we not need to solve the MVMP, but in many cases we won't even need to know M^b (or M^a).

Based on what has been established so far, we show how the theory of Krylov subspaces can be used to construct a solution to the MPP.

2.2 Preliminaries and Krylov subspaces

For arbitrary $\mathbf{v} \in (\mathbb{Z}_p^n)^*$ and $M \in GL(n,p)$, the minimal polynomial of M relative to $\mathbf{v}$, $\mathcal{P}_{\min}^{\mathbf{v},M}$, is the monic polynomial of smallest degree such that $\mathbf{v}\mathcal{P}_{\min}^{\mathbf{v},M}(M) = \mathbf{0}$. Moreover, $\mathcal{P}_{\min}^{\mathbf{v},M}$ divides the minimal polynomial of M, $\mathcal{P}_{\min}^{M}$, and thus $k = \deg \mathcal{P}_{\min}^{\mathbf{v},M} \leq \deg \mathcal{P}_{\min}^{M} = m \leq n$ (see [DPW05, Section 2]). Let

$$\mathcal{K}_{\mathbf{v},M} = \operatorname{span}\{\mathbf{v}, \mathbf{v}M^1, \mathbf{v}M^2, \dots\}. \tag{5}$$

From $\mathbf{v}\mathcal{P}_{\min}^{\mathbf{v},M}(M) = \mathbf{0}$ and the minimality of $\mathcal{P}_{\min}^{\mathbf{v},M}$, it follows that $\{\mathbf{v}, \mathbf{v}M^1, \dots, \mathbf{v}M^{k-1}\}$ is a basis of $\mathcal{K}_{\mathbf{v},M}$. Similarly, from $\mathcal{P}_{\min}^{M}(M) = 0_{n\times n}$ and the minimality of $\mathcal{P}_{\min}^{M}$ we conclude $\{M^0, M^1, \dots, M^{m-1}\}$ is a basis of $\mathcal{V}$ (the linear space defined at the beginning of Section 2.1).

2.3 The attack

Following the definitions present in §1.4, for an instance of the MKE's public parameters $\mathbf{pub} = (n, p, M, \mathbf{v}, \mathbf{w}_A, \mathbf{w}_B)$, we have $\mathbf{w}_B = \mathbf{v}M^b$ for some $b \in \mathbb{Z}$. Therefore $\mathbf{w}_B \in \mathcal{K}_{\mathbf{v},M}$ and there exist coefficients $c_0, \dots, c_{k-1} \in \mathbb{Z}_p$ such that

$$\mathbf{w}_B = c_0\mathbf{v}M^0 + \dots + c_{k-1}\mathbf{v}M^{k-1} = \mathbf{v}(c_0M^0 + \dots + c_{k-1}M^{k-1}).$$

The matrix $c_0M^0 + \dots + c_{k-1}M^{k-1}$ is thus a solution to the MPP. The algorithm below combines this fact with the basis of $\mathcal{K}_{\mathbf{v},M}$ described after (5) to construct an attack on the MKE protocol:

Algorithm 1 Key-recovery algorithm

Input: **pub** — MKE public parameters
Output: $\mathbf{w}_K = \mathbf{v}M^{a+b}$

1: Use a suitable Krylov basis algorithm to determine the dimension k of $\mathcal{K}_{\mathbf{v},M}$ and the Krylov basis matrix

$$K \leftarrow \left(\begin{array}{c|c|c|c} \mathbf{v}^T & (\mathbf{v}M^1)^T & \dots & (\mathbf{v}M^{k-1})^T \end{array} \right)$$

2: Solve for $\mathbf{c} = (c_0, \dots, c_{k-1})$ the linear system $K\mathbf{c}^T = \mathbf{w}_B^T$
3: **return** $\mathbf{w}_A \cdot (c_0 I + c_1 M + \dots + c_{k-1}M^{k-1})$

Theorem 1. *The key-recovery Algorithm 1 correctly outputs the secret* $\mathbf{w}_K$.

Proof. The $n \times k$ matrix K has columns that form a basis for $\mathcal{K}_{\mathbf{v},M}$. We know there is some $b \in \mathbb{Z}$ such that $\mathbf{w}_B = \mathbf{v}M^b$ so $\mathbf{w}_B \in \mathcal{K}_{\mathbf{v},M} = \mathrm{Im}(K)$ and the linear system in step (2) must have an unique solution $\mathbf{c} = (c_0, \ldots, c_{k-1})$.

Now the computation in the last step yields:

$$\begin{aligned}
\mathbf{w}_A(c_0 I + c_1 M + \ldots + c_{k-1}M^{k-1}) &= \mathbf{v}M^a(c_0 I + c_1 M + \ldots + c_{k-1}M^{k-1}) \\
&= \mathbf{v}(c_0 I + c_1 M + \ldots + c_{k-1}M^{k-1})M^a \\
&= (c_0\mathbf{v} + c_1\mathbf{v}M + \ldots + c_{k-1}\mathbf{v}M^{k-1})M^a \\
&= \mathbf{c}K^T M^a \\
&= \mathbf{w}_B M^a \\
&= \mathbf{w}_K.
\end{aligned}$$

□

Theorem 2. *There is an instance of the key-recovery Algorithm 1, using classic matrix multiplication, that does $\mathcal{O}(n^2 k)$ algebraic operations.*

Proof. We will detail the cost of every step performed:

Step 1: As noted in [DPW05, Section 2.1], in the `MinPoly` algorithm the creation of the Krylov matrix can be merged with its LSP factorization and early-terminated once the rank reaches k. This yields a procedure that constructs the first k Krylov vectors together with the LSP factorization of the Krylov matrix using $\mathcal{O}(n^2 k)$ operations under classical matrix arithmetic; [1]

Step 2: Since K^T is full row-rank, its LSP factorization coincides with its LUP factorization. So now the system in step 2 can be expressed as $P^T U^T L^T \mathbf{c}^T = \mathbf{w}_B^T$ where P^T is an $n \times n$ permutation matrix, U^T is an $n \times k$ lower trapezoidal matrix with non-zero elements on the main diagonal and L^T is a $k \times k$ unit upper triangular matrix. With the aid of variable vectors $\mathbf{x}_{1\times n}$ and $\mathbf{y}_{1\times k}$, we can break this problem into three smaller subproblems:

Compute (a) $\mathbf{x}^T = P\mathbf{w}_B^T$ then solve (b) $U^T\mathbf{y}^T = \mathbf{x}^T$ and finally (c) $L^T\mathbf{c}^T = \mathbf{y}^T$. Following [AHU74, Chapter 6] we can conveniently represent P as an array of n elements, which allows solving (a) in $\mathcal{O}(n)$ operations. The subproblems (b) and (c) can both be solved in $\mathcal{O}(k^2)$ operations by doing, respectively, forward and backward substitution. Thus, by using the LUP factorization, this system can be solved in $\mathcal{O}(n + k^2)$ operations.

Step 3: The trick here is to avoid doing matrix products. Compute $\mathbf{w}_K$ as a linear combination of vector-matrix products using $\mathcal{O}(n^2 k)$ operations using the algorithm bellow:

Algorithm 2 Step 3 subAlgorithm

```
auxVec ← w_A
w_K ← c_0 · w_A
for i = 1 to k − 1 do                    ▷ O(k) iterations
    auxVec ← auxVec · M                  ▷ O(n^2) operations
    w_K ← w_K + c_i · auxVec
end for
return w_K
```

The dominant term in steps 1, 2 and 3 is $\mathcal{O}(n^2 k)$. □

[1] Using fast matrix arithmetic, for some $\omega \in [2, 3)$, the matrix K can be computed in $\mathcal{O}(n^\omega \log k)$ in the `MinPoly` or $\mathcal{O}(n^\omega)$ if we use the `MaxKrylovBasis` algorithm [NPV24, Chapter 4]. In the second case, we don't get the LUP factorization of K^T but we could solve the linear system in step 2 using row reduction.

The attack delineated in [Meg14] becomes a particular instance of this algorithm when $k = n$. In this case, as it will become clear from next subsection, $M^b = c_0 I + c_1 M + \ldots + c_{k-1} M^{k-1}$. This explain why the example presented in that article works.

2.4 On the diversity of solutions

The MPP may have multiple solutions. Taking, for instance, $\mathbf{v} = (1, 0, 0, \ldots, 0)$, $M = \text{diag}(2, 1, 1, \ldots, 1)$, $b = 5$ and $p = 41$ leads to simple examples like $32M^0$ and $6M^2 + M^3$, which are different matrices and not just the same matrix being expressed as distinct polynomials in M.

Note that each vector $(c_0, \ldots, c_{m-1})^T$ that solves the below linear system corresponds to an unique solution $c_0 M^0 + \ldots + c_{m-1} M^{m-1}$ to the MPP and vice-versa:

$$\left(\begin{array}{c|c|c|c} \mathbf{v}^T & (\mathbf{v}M^1)^T & \ldots & (\mathbf{v}M^{m-1})^T \end{array} \right) \cdot \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \end{pmatrix} = \mathbf{w}_B^T.$$

The linear system in step (2) of the key-recovery algorithm (in Section 2.3) has a single solution, which means it characterizes the unique matrix X (polynomial in M of degree $< k$) such that $\mathbf{v}X = \mathbf{w}_B$. We conclude $X = M^b \iff M^b \in \text{span}\{M^0, \ldots, M^{k-1}\}$. We also present the following fact that is important enough to have its own theorem:

Theorem 3. *The solution to the MPP is unique if, and only if,* $\mathcal{P}_{\min}^{M} = \mathcal{P}_{\min}^{\mathbf{v},M}$. *More generally, the set of solutions is an affine space with dimension* $\deg \mathcal{P}_{\min}^{M} - \deg \mathcal{P}_{\min}^{\mathbf{v},M}$.

Proof. Let $\mathcal{V}$ be as defined in Subsection 2.1, the parameters m and k as defined in Subsection 2.2 and Λ the subspace consisting of matrices $\mathcal{P}_\Lambda(M) \in \mathcal{V}$ such that $\mathbf{v}\mathcal{P}_\Lambda(M) = \mathbf{0}$.

For any $\mathcal{P}_\Lambda(M) \in \Lambda$, $\mathbf{v}(M^b + \mathcal{P}_\Lambda(M)) = \mathbf{v}M^b + \mathbf{v}\mathcal{P}_\Lambda(M) = \mathbf{w}_B$, and for any $\mathcal{P}_\alpha(M) \in \mathcal{V} \setminus \Lambda$, $\mathbf{v}(M^b + \mathcal{P}_\alpha(M)) \neq \mathbf{w}_B$. So $\{X \in \mathcal{V} | \mathbf{v}X = \mathbf{w}_B\}$ is the affine subspace $M^b + \Lambda$. As such, the solution to the MPP is unique if, and only if, Λ is 0-dimensional.

Consider the linear transformation $L_{\mathbf{v}} : \mathcal{V} \to \mathcal{K}_{\mathbf{v},M}$ defined by $L_{\mathbf{v}}(\alpha_0 M^0 + \ldots + \alpha_{m-1} M^{m-1}) = \alpha_0 \mathbf{v} M^0 + \ldots + \alpha_{m-1} \mathbf{v} M^{m-1}$. $L_{\mathbf{v}}$ is clearly surjective and by the rank-nullity theorem the dimension of the kernel of $L_{\mathbf{v}}$, Λ, is $m - k$. □

An immediate consequence of the above theorem is there are $p^{(m-k)}$ distinct solutions to the MPP.

3 Conclusions

Based on our results, we conclude the MKE is insecure for any choice of parameters, even against a classical adversary, as the secret shared key $\mathbf{w}_K$ can be efficiently recovered from the public parameters. Even if we assume recovering M^a or M^b is difficult, we can still efficiently recover a matrix solution to the MPP.

In this article we have improved upon previous attacks. Arzaki's collision algorithms [AW16] and his diagonalization technique [AMP18], when applicable, recover $\mathbf{w}_K$ in sub-exponential time in n. Megrelishvili's [Meg14] idea leads to a polynomial time attack, but only works when $\deg \mathcal{P}_{\min}^{\mathbf{v},M} = n$ as it requires the linear independence of $\{\mathbf{v}, \mathbf{v}M, \ldots, \mathbf{v}M^{n-1}\}$, and, when applicable, it is a particular case of our approach.

We have reduced the recovery of the shared secret key $\mathbf{w}_K$ to a new problem, the Matrix Polynomial Problem, making the MKE substantially easier to break than it was before. The key idea to build an algorithm that generates a matrix polynomial in M that

maps $\mathbf{v}$ to $\mathbf{w}_B$ generated, to the best of our knowledge, the strongest attack in literature against the MKE, as it recovers $\mathbf{w}_K$ in polynomial time for any instance of the protocol. Interestingly, we attack the MKE analogue of the Computational DH Problem directly, without trying to recover the secret values b or M^b.

It also seems likely based on the results of this article that, in the worst-case scenario, recovering $\mathbf{w}_K$ is in fact much easier than recovering a, b or even M^a or M^b. Which would imply the MKE is a Diffie-Hellman-like protocol that does not have the desired equivalence between its computational problem of deriving the shared secret key and the problem of deriving the private secret key chosen by Alice or by Bob.

We have not provided in this article an efficient algorithm that can always recover the matrix powers M^b or M^a from **pub**. That remains an open problem, so it is possible some similar or related protocols remain secure for now if adequate parameters are chosen. We have however delimited a search space for these matrices and quantified this space's size (p^{m-k} from Theorem 3).

Acknowledgments We would like to thank the anonymous reviewers for their insightful comments and constructive suggestions, which significantly improved the quality and clarity of this manuscript.

References

[AHU74] Alfred V Aho, John E Hopcroft, and Jeffrey D Ullman. *The design and analysis of computer algorithms*. Addison-Wesley Publishing Company, 1974.

[AMP18] Muhammad Arzaki, Danang Triantoro Murdiansyah, and Satrio Adi Prabowo. Breaking megrelishvili protocol using matrix diagonalization. In *Journal of Physics: Conference Series*, volume 971. IOP Publishing, 2018. doi:10.1088/1742-6596/971/1/012025.

[Arz16] Muhammad Arzaki. Elementary algorithms analysis of megrelishvili protocol. *Indonesia Journal on Computing (Indo-JC)*, 1(1):11–23, 2016. doi:10.21108/INDOJC.2016.1.1.52.

[Arz17] Muhammad Arzaki. On the generalizations of megrelishvili protocol for group key distribution. *Indonesian Journal on Computing (Indo-JC)*, 2(2):55–78, 2017. This paper is the likely published version of the 'Extending Megrelishvili protocol for multi-party key agreement' file. doi:10.21108/INDOJC.2017.2.2.179.

[Arz18] Muhammad Arzaki. Strengthening megrelishvili protocol against man-in-the-middle attack. In *2018 6th International Conference on Information and Communication Technology (ICoICT)*, pages 274–280. IEEE, 2018. doi:10.1109/ICoICT.2018.8528731.

[Arz19] Muhammad Arzaki. Constructing a one-time and a few-time digital signature schemes from the hardness of megrelishvili vector-matrix problem. In *2019 7th International Conference on Information and Communication Technology (ICoICT)*, pages 1–8. IEEE, 2019. doi:10.1109/ICoICT.2019.8835378.

[AW16] Muhammad Arzaki and Bambang Ari Wahyudi. Collision algorithms for breaking megrelishvili protocol: Theory and numerical experiments. In *2016 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, pages 53–60. IEEE, 2016. doi:10.1109/ICACSIS.2016.7872728.

[AW17] Muhammad Arzaki and Bambang Ari Wahyudi. Extending megrelishvili protocol for multi-party key agreement. In *2017 5th international conference on information and communication technology (ICoIC7)*, pages 1–8. IEEE, 2017. doi:10.1109/ICoICT.2017.8074690.

[Chi17] Andrew M Childs. Lecture notes on quantum algorithms. *Lecture notes at University of Maryland*, 5, 2017. Accessed: 2025-07-07. URL: https://www.cs.umd.edu/~amchilds/qa/qa.pdf.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. doi:10.1109/TIT.1976.1055638.

[DPW05] Jean-Guillaume Dumas, Clément Pernet, and Zhendong Wan. Efficient computation of the characteristic polynomial. In *Proceedings of the 2005 international symposium on Symbolic and algebraic computation*, pages 140–147, 2005. doi:10.1145/1073884.1073905.

[IWI+21] A Haj Ismail, Hafiz Muhammad Waseem, Muhammad Ishtiaq, Sajjad Shaukat Jamal, and Majid Khan. Quantum spin half algebra and generalized megrelishvili protocol for confidentiality of digital images. *International Journal of Theoretical Physics*, 60(5):1720–1741, 2021. doi:10.1007/s10773-021-04794-0.

[MCC06] R Megrelishvili, M Chelidze, and K Chelidze. On the construction of secret and public-key cryptosystems. *Appl. Math., Inform. Mech*, 11(2):29–36, 2006.

[Meg14] Richard P Megrelishvili. Analysis of the matrix one-way function and two variants of its implementation. *Computer Science & Telecommunications*, 43(3), 2014.

[MW97] A. J. Menezes and Y.-H. Wu. The discrete logarithm problem in GL(n, q). *Ars Combinatoria*, 47:23–32, 1997. URL: https://combinatorialpress.com/article/ars/Volume%20047/volume_47_paper-3.pdf.

[NPV24] Vincent Neiger, Clément Pernet, and Gilles Villard. Computing krylov iterates in the time of matrix multiplication. In *Proceedings of the 2024 International Symposium on Symbolic and Algebraic Computation*, pages 419–428, 2024. doi:10.1145/3666000.3669715.

[Sho94] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994. doi:10.1109/SFCS.1994.365700.

[WJHK21] Hafiz Muhammad Waseem, Sajjad Shaukat Jamal, Iqtadar Hussain, and Majid Khan. A novel hybrid secure confidentiality mechanism for medical environment based on kramer's spin principle. *International Journal of Theoretical Physics*, 60(1):314–330, 2021. doi:10.1007/s10773-020-04694-9.

Journal of the Brazilian Computer Society, 2026, 32:1, doi: 10.5753/jbcs.2026.5486
This work is licensed under a Creative Commons Attribution 4.0 International License.

Lattice Basis Reduction Attack on Matrix NTRU

Thiago do Rego Sousa [**CEPESC** | ***thiagodoregosousa@gmail.com***]
Tertuliano Carneiro de Souza Neto [**CEPESC** | ***tsouzaneto@gmail.com***]

Research and Development Center for Communication Security (CEPESC), Brasília, DF, 70610-905, Brazil.

Received: 14 February 2025 • **Accepted:** 21 November 2025 • **Published:** 24 April 2026

Abstract NTRU is one of the most important post-quantum cryptosystems nowadays and since its introduction several variants have been proposed in the literature. In particular, the Matrix NTRU is a variant which replaces the NTRU polynomials by integer matrices. In this work, we develop a lattice-based reduction attack on the Matrix NTRU cryptosystem that allows us to recover the plaintext. We also show that this system is completely vulnerable to the proposed attack for parameters that could be used in practice. We show that this practical attack can also be extended by reducing the lattice dimension. In addition, we give sufficient conditions to avoid decryption failure for the Matrix NTRU.

Keywords: Post-quantum cryptography, NTRU, Matrix NTRU, Lattice cryptanalysis, Lattice attack, Decryption failure, Cryptanalysis

1 Introduction

Quantum computers are now a reality and if large-scale ones are built, they could break most of the public key cryptosystems used currently. That is one of the reasons that research on post-quantum systems (cryptographic systems underpinned on mathematical problems that are intractable by both quantum and conventional computers) has taken off in the last decades.

In 2016, the National Institute of Standards and Technology (NIST) started a public process for the standardization of post quantum algorithms that could be used to interoperate with existing communications protocols and networks currently in use. One of the main mathematical problem underpinning such system is the shortest vector problem (SVP), which aims at finding a shortest vector in a structure called lattice (see Silverman *et al*. [2008]). Two lattice-based systems have progressed significantly in the NIST Post-Quantum Cryptography standardization process, namely NTRU Chen *et al*. [2020a] and NTRU Prime Bernstein *et al*. [2017], both based on the original NTRU system from Hoffstein *et al*. [1998].

The NTRU system was presented in 1996 to the cryptographic community during the CRYPTO'96 rump session. Since its introduction, the NTRU system has been extensively analyzed and extended in many different ways. Interesting review articles containing a wealth of examples and references include Singh and Padhye [2016]; Salleh and Kamarulhaili [2020] and Mittal and Ramkumar [2022]. NTRU is based on modular algebra of polynomials in specific rings and such generalizations range from changing the polynomial coefficients to using matrix structures.

The first NTRU generalization using matrices was introduced in Coglianese and Goi [2005], where key generation, encryption and decryption operate over matrices whose entries are polynomials. In 2008, Nayak *et al*. [2008] proposed a matrix-based system as a variant of NTRU using only matrices with integer entries, replacing the arithmetic of truncated polynomials with a simple modular arithmetic on matrices. This new system, called Matrix NTRU, was subject to some further analysis and even suggested for real data applications. Indeed, Kumar *et al*. [2013] presented a framework for deploying Matrix NTRU in real data applications. More recently, Wijayanti *et al*. [2023a] developed a variant of the Matrix NTRU which is based on matrix arithmetic defined over $\mathbb{Z}[\sqrt{-3}]$ and do Rêgo Sousa and Neto [2025] improved the key generation and corrected the decryption failure bound.

1.1 Related works

Since its introduction the Matrix NTRU has attracted some attention. A comparative study regarding speed and key sizes was performed in Nayak *et al*. [2012] showing that Matrix NTRU was faster than the classical NTRU for short dimensions. In addition, Nayak *et al*. [2012] argue that Matrix NTRU was more secure than the classical NTRU since matrices are non-commutative. However, we will show that the Matrix NTRU system is seriously affected by the lattice attack developed in Section 5. This demonstrates that the Matrix NTRU is far weaker than the classical NTRU system for comparable key sizes, contrary to what was accredited in the literature (see Nayak *et al*. [2012] and Kumar *et al*. [2013]).

Subsequent works on the Matrix NTRU focused on expanding the key space and on eliminating restrictions on the matrices representing the public key by using Gram-Schmidt orthogonalization and companion matrices (Luo and Lin [2011]; Tripathi *et al*. [2016]; Mamdikar *et al*. [2013]).

With regards to the system's security, Nayak *et al*. [2011] applied a reaction type attack to Matrix NTRU, by adapting the results of the original attack presented in Hall *et al*. [1999] for the NTRU system. In addition, a meet in the middle attack was recently presented in Wijayanti *et al*. [2023b]. However, the simulation studies presented in these works only show the performance of the aforementioned attacks for very low

dimensions of the Matrix NTRU, which cannot be used in practice.

Our main contribution to Matrix NTRU is in developing a lattice-based attack and analyse its resistance in dimensions much higher than the ones suggested for practical applications in Kumar *et al*. [2013]. We will show in Section 5 how this attack can recover the private key (up to a permutation). In addition, this attack is further used to construct a message recovery attack that works for a dimension of the system of up to 110. According to Nayak *et al*. [2012], a Matrix NTRU system such as this is equivalent to an NTRU system using polynomials of degree 110^2, which is remains far from being broken by lattice-based techniques (Chen *et al*. [2020a]). This attack is further improved by using the dimension reduction ideas of May and Silverman [2001] leading to practical attacks for $n = 120$.

In addition to the lattice attack, we also analyze the decryption failure rate of the Matrix NTRU. The results of Proposition 2 give sufficient conditions that depend only on the system's public parameters to avoid decryption failure.

1.2 Structure of the paper

The remainder of the paper is organized as follows. In Section 2, we describe the NTRU cryptosystem in detail, and a brief description of Matrix NTRU is given in Section 3. Sufficient conditions to avoid decryption failure for the Matrix NTRU are also presented in this section. Then, in Section 4, we construct the lattice structure associated with the Matrix NTRU and report the results of an attack to recover the private key. This attack is further refined in Section 5 to construct a message recovery type attack that works for a dimension of up to $n^2 = 110^2$ on a personal computer. Finally, Section 6 improves the attack by using the dimension reduction techniques of May which allowed us to run it successfully for $n = 120$ on a personal computer. The results of the previous section are summarized in Section 7 where we explain why the Matrix NTRU is far weaker than the NTRU cryptosystem concerning the lattice attack.

2 The NTRU Cryptosystem

Let n and p be prime numbers. Let $q \geq 1$ be an integer such that $(n, q) = (p, q) = 1$. The main arithmetic operations in the NTRU cryptosystem are calculations over polynomials defined over the rings R, R_p e R_q defined as:

$$R = \frac{\mathbb{Z}[x]}{(x^n - 1)}, R_p = \frac{(\mathbb{Z}/p\mathbb{Z})[x]}{(x^n - 1)}, R_q = \frac{(\mathbb{Z}/q\mathbb{Z})[x]}{(x^n - 1)}.$$

We can notice that the ring R is related to the other two. In fact, for every $a(x)$ in R, we can identify it to an element in R_p or R_q by applying reduction modulo p or q, respectively, over the coefficients of $a(x)$. We say that $a(x)$ is a ternary polynomial if its coefficients lie in the set $\{-1, 0, 1\}$.

The NTRU cryptosystem works as follows.

1. Key Generation: Alice chooses two ternary polynomials at random, namely, $f(x)$ and $g(x)$. Next, Alice tries to compute the inverse of f over the rings R_q, R_p, i.e, $f_q(x) = f(x)^{-1} \in R_q$ and $f_p(x) = f(x)^{-1} \in R_p$ until she succeeds. Finally, she computes the polynomial
$$h(x) = f_q(x) * g(x) \in R_q,$$
where $*$ denotes polynomial multiplication in R_q, i.e., a cyclic convolution product as defined in [Hoffstein *et al*., 1998, Section 1.1] The polynomial $h(x)$ is Alice's public key and the pair $(f(x), f_p(x))$ is her private key.
2. Encryption: Suppose Bob wants to send an encrypted message to Alice and let $m(x) \in R_p$ be Bob's plaintext. Next, Bob chooses, at random, a ternary polynomial $r(x)$ and computes
$$e(x) \equiv ph(x) * r(x) + m(x) \mod q.$$
Notice that Bob's ciphertext $e(x)$ is an element of the ring R_q.
3. Decryption: Once Alice receives Bob's ciphertext $e(x)$, she starts the decryption process by computing
$$a(x) \equiv f(x) * e(x) \mod q.$$
Then the reduction modulo p gives the desired plaintext
$$b(x) \equiv f_p(x) * a(x) \mod p.$$

Because of the randomness of the polynomial $r(x)$, NTRU is a probabilistic cryptosystem, since a message $m(x)$ can be encrypted to several ciphertexts $ph(x) * r(x) + m(x)$, depending on each instance of $r(x)$. In addition to that, during decryption the coefficients of the polynomial $a(x)$ can not grow deliberately because that would invalidate the computations modulo p that follows, a phenomenon known as a decryption failure. Some attacks in the literature take advantage of these decryption failures Howgrave-Graham *et al*. [2003]; Gama and Nguyen [2007]; Jaulmes and Joux [2000] and therefore we should choose the parameters carefully.

3 The Matrix NTRU Cryptosystem

Let p be a prime number and $q >> p$ an integer such that $(p, q) = 1$. Let

$$A = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix}$$

be an $n \times n$ matrix whose entries are integer numbers, that is, $A \in M_n(\mathbb{Z})$. In a similar way we have done before with polynomials, we say that A is a ternary matrix if all of its entries lie in the set $\{-1, 0-, 1\}$.

We say that A is reduced modulo p, denoted as $A \mod p$, if every entry of A is reduced modulo p. Therefore

$$A \mod p = \begin{pmatrix} a_{11} \mod p & \dots & a_{1n} \mod p \\ \vdots & \ddots & \vdots \\ a_{n1} \mod p & \dots & a_{nn} \mod p \end{pmatrix}.$$

Notice that, in the latter case, we can view matrix A as an element of the ring $M_n(\mathbb{F}_p)$. In this work, we are going to consider the center lift operation modulo p and q in the

decryption process. In that case, it can be useful to consider a non-canonical representation of the elements on the rings. Therefore, when we are dealing with a prime modulus, we have

$$\mathbb{F}_p = \left\{-\frac{p-1}{2}, \ldots, \frac{p-1}{2}\right\}$$

and for a composite number, we have

$$\mathbb{Z}_q = \left\{-\frac{q}{2}+1, \ldots, \frac{q}{2}\right\}.$$

Modular arithmetic over the rings $M_n(\mathbb{F}_p)$ and $M_n(\mathbb{Z}_q)$ is what underpins the Matrix NTRU system. In such rings we can add, subtract and multiply matrices in the same way we have done with matrices over the field of real numbers. However, in order to invert a matrix in $M_n(\mathbb{F}_p)$ we have to be sure that p and the determinant of A are relatively prime Jacques-García *et al*. [2022]; Wijayanti *et al*. [2023b]. If that is the case, then there exists a (unique) matrix B such that $AB = BA = I$ in $M_n(\mathbb{F}_p)$, where I is the identity matrix. The matrix B is called the inverse of A modulo p and denoted $A^{-1} \mod p$.

The Matrix NTRU system works as follows.

1. Key generation: The private and public keys are $n \times n$ matrices. First, we choose a pair of ternary matrices F, G such that $F \mod p$ and $F \mod q$ are invertible in $M_n(\mathbb{F}_p)$ and $M_n(\mathbb{Z}_q)$, respectively. Then we compute the matrices F_p and F_q, where $F_p = F^{-1} \mod p$ and $F_q = F^{-1} \mod q$. The pair F, F_p is the private key and the parameters F, G, F_p, F_q should be kept in secret. Now, we can compute the public key by performing the calculation
$$H = pF_qG \mod q.$$
2. Encryption: To encrypt a message, we encode it as a matrix $M \in M_n(\mathbb{F}_p)$ and choose a ternary matrix $R \in M_n(\mathbb{Z})$, at random. Now, we compute the ciphertext as
$$E = HR + M \mod q.$$
3. Decryption: To decrypt, we compute
$$A = FE \mod q,$$
where F is the private key we saw previously. Under simple conditions on the system parameters (see Proposition 1 bellow), reducing A modulo p gives the desired plaintext
$$B = F_pA \mod p$$

Depending on the parameters selection, we can have errors when operating on Matrix NTRU, just as we have when dealing with NTRU cryptosystem. Sometimes, the matrices B and M can be different and we say there is a decryption failure if that is the case. We notice that the authors Nayak *et al*. [2008] did not address that question. However, it turns out that this is an important issue concerning any cryptosystem since attackers can take advantage of it. To fill in this gap, we prove the following result, which shows that there are parameter selections to avoid decryption failure on Matrix NTRU.

Proposition 1. *Suppose that p, q and n are fixed parameters for the Matrix NTRU defined above. If $n(3p-1) < q$, then any ciphertext E resulting from the encryption of a message M with private key F, decrypts correctly to M.*

Proof. During decryption, one computes

$$\begin{aligned} A &= FE \mod q \\ &= F(HR+M) \mod q \\ &= FHR + FM \mod q \\ &= pFF_qGR + FM \mod q \\ &= pGR + FM \mod q. \end{aligned} \tag{1}$$

Following the same arguments as in the proof of Proposition 6.48 in Silverman *et al*. [2008], we need to bound the largest coefficient (in modulus) of each entry of the matrix $pGR + FM$ computed exactly (without module q). For decryption to work, the above matrix should have entries whose absolute value does not exceed $q/2$. Since every entry of the matrices G and R lie in $\{-1, 0, 1\}$, the largest possible entry of the product GR is upper bounded by n. On the other hand, the matrix M has coefficients between $-(p-1)/2$ and $(p-1)/2$ and F has coefficients lying in $\{-1, 0, 1\}$. Therefore, all coefficients of the product FM can be bounded by $n(p-1)/2$. Finally, the entries of the matrix $pGR + FM$ are all upper bounded in module by $np + n(p-1)/2 = n(3p-1)/2$. Under the assumption of the proposition, this can be further bounded by $q/2$ which will ensure correct decryption of the ciphertext. □

We implemented the Matrix NTRU in `sagemath` (Team [2024]) and assessed the decryption failure rate for a set of different parameters. Table 1 shows the proportion of ciphertext that could not be decrypted correctly, over 10000 different replications of the process for varying n and q. Under the conditions of Proposition 1, we avoid decryption failure whenever $n(3p-1) < q$, and the results for the parameters satisfying these conditions are shown by a gray strip in Table 1. One important fact about Proposition 1 is that as n grows, q should also grow, otherwise one might have decryption failures in the Matrix NTRU system. A similar situation also happens in the classical NTRU system (see [Silverman *et al*., 2008, Proposition 6.48])

4 Lattice attack on the private key

In the following section we show how the problem of finding the private key F of the Matrix NTRU system is connected to solving a well-known problem in lattices, namely finding a shortest vector in a lattice. A lattice can be defined in the following way.

Definition 1. *([Silverman* et al*., 2008, Section 6.4]) Let $f_1, \ldots, f_n \in \mathbb{R}^n$ be a set of linearly independent vectors. The lattice L generated by $f_1, \ldots, f_n$ is the set of linear combinations of $f_1, \ldots, f_n$ with coefficients in $\mathbb{Z}$,*

$$L = \{\alpha_1 f_1 + \cdots + \alpha_n f_n : \alpha_1, \ldots, \alpha_n \in \mathbb{Z}\}. \tag{2}$$

Table 1. Decryption failure estimated probability for the Matrix NTRU system for fixed $p = 3$ and varying n and q. For each parameter setting a new key and message were generated and encrypted. Results present the proportion of messages decrypted INCORRECTLY over 10 000 repetitions. Gray strips represent experimental decryption failure rate where the conditions of Proposition 1 are met.

	q										
n	32	64	79	128	256	307	512	701	1024	2048	4096
5	0.008	0	0	0	0	0	0	0	0	0	0
10	0.775	0	0	0	0	0	0	0	0	0	0
20	1	0.210	0.008	0	0	0	0	0	0	0	0
30	1	0.989	0.397	0	0	0	0	0	0	0	0
40	1	1	0.989	0.002	0	0	0	0	0	0	0
50	1	1	1	0.039	0	0	0	0	0	0	0
60	1	1	1	0.257	0	0	0	0	0	0	0
70	1	1	1	0.734	0	0	0	0	0	0	0
80	1	1	1	0.984	0	0	0	0	0	0	0
90	1	1	1	1	0	0	0	0	0	0	0
100	1	1	1	1	0	0	0	0	0	0	0
110	1	1	1	1	0	0	0	0	0	0	0

The set of vectors $f_1, \ldots, f_n$ is called the lattice basis and it is usual to stack them into a matrix and work with the matrix as being the lattice basis that generates L. A fundamental problem in lattice is finding a shortest nonzero vector in the lattice which minimizes the Euclidean norm $\|f\|$. This is called the shortest vector problem (SVP). It is important to notice that the SVP problem asks for a shortest vector and not the shortest vector since e.g. f and $-f$ have the same Euclidean norm. According to Silverman *et al*. [2008] finding a solution for the SVP problem can be used to break various cryptosystem, in particular the NTRU system from Section 2 (for certain parameters) and as we will show next how it can be used to finding the private key F in the Matrix NTRU system. It is worth noticing that the current version of NTRU submitted to NISTs post quantum competition Chen *et al*. [2020a] have parameters that avoid private key attacks with current computational resources and it is not practical. On the other hand, the Matrix NTRU variant has a serious vulnerability in its construction, that makes it breakable for quite high values of parameters that could be used in practice.

Proposition 2. *Let F, G be the private keys and let H be the public key of the Matrix NTRU with parameters n, p, q. Let (f_k, g_k) be the k-th line of F and G respectively and p^{-1} be the inverse of p module q. Then, (f_k, g_k) belongs to the lattice generated by the lines of the $2n \times 2n$ block matrix*

$$L = \begin{pmatrix} I_n & p^{-1}H \\ 0_n & qI_n \end{pmatrix}, \tag{3}$$

where I_n is the n dimensional identity matrix and 0_n is the n dimensional zero matrix. In other words, the unknown vector (f_k, g_k) can be written as an integer linear combination of the lines of L (which has only known quantities).

Proof. The public key is defined as $H = pF_qG$, where p and q are coprime, and F and G are the private keys created during the key generation step. Multiplying both sides of the above equation by Fp^{-1} we get $Fp^{-1}H = G \mod q$. This implies that there exists a matrix $A \in M(\mathbb{Z})$ such that

$$G = F(p^{-1}H) + qA, \tag{4}$$

Let $V = (\alpha H)$ and denote by f_k, g_k and a_k the k-th line of the matrices F, G and A. Equation (4) implies that $g_k = f_kV + qa_k$. Therefore, the $1 \times 2n$ vector (f_k, g_k) can be written as

$$\begin{aligned}(f_k, g_k) &= (f_k, f_kV + a_k(qI_n)) \\ &= (f_kI_n + a_k0_n, f_kV + a_k(qI_n)) \\ &= f_k(I_n, V) + a_k(0_n, qI_n) \\ &= f_{k1}(1, 0, \ldots, 0, 0, v_{11}, v_{12}, \ldots, v_{1n}) \\ &+ f_{k2}(0, 1, \ldots, 0, 0, v_{21}, v_{22}, \ldots, v_{2n}) \\ &\ldots \\ &+ f_{kn}(0, 0, \ldots, 0, 1, v_{n1}, v_{n2}, \ldots, v_{nn}) \\ &+ a_{k1}(0, 0, \ldots, 0, 0, q, 0, \ldots, 0) \\ &+ a_{k2}(0, 0, \ldots, 0, 0, 0, q, \ldots, 0) \\ &\ldots \\ &+ a_{kn}(0, 0, \ldots, 0, 0, 0, 0, \ldots, q).\end{aligned} \tag{5}$$

Since all $1 \times 2n$ dimensional vectors at the rhs of (5) equation are exactly the lines of the matrix L in (3), the proof is completed. □

Recall that the matrices F and G have only coefficients in $\{-1, 0, 1\}$, and therefore any line of the type (f_k, g_k) is a relatively short vector in the lattice L for large q. Indeed, using a Gaussian heuristic, the shortest vector expected from a lattice L depends only on the dimension of L and its determinant. Since L is upper triangular, we have $\det(L) = q^n$. Therefore, the length of the shortest vector expected from L is

$$l = \sqrt{\frac{n}{2\pi e}}(\det(L))^{1/(2n)} = \sqrt{\frac{nq}{2\pi e}}.$$

The target vector (f_k, g_k) has varying norm depending on how the private key is chosen. If every entry is chosen with uniform probability on $\{-1, 0, 1\}$, then, its expected norm is $\alpha = \sqrt{2n/3}$. This means that for higher values of q, the target vector has norm smaller than the expected by the by the Gaussian heuristics, which means that the LLL algorithm has a high probability of find a short vector in L (see Silverman *et al*. [2008]).

Another interesting fact is that we do not need to attack the whole private key F which has dimension n^2. Using a suitable algorithm to solve the SVP in L we can try to find any line of F and hopefully all lines separately reducing the complexity of the attack by a factor of n.

Solving the SVP problem can be done in many ways, and one commonly done is by using lattice-based reduction algorithms such as the famous LLL algorithm from Lenstra *et al*. [1982] and or the BKZ algorithms Chen and Nguyen [2011]. For more details on the LLL algorithm see e.g. [Bremner, 2011, Chapter 4]. These base reduction algorithms try to find a shorter basis for the lattice L, and in doing so, they usually return potential short vectors for the lattice L and we can test if they correspond to any line of the matrix F.

In addition to using the LLL algorithm, there is an improved variant of lattice based reduction algorithms called BKZ as defined in Chen and Nguyen [2011]. The BKZ algorithm, proceeds with repeated local improvements to the lattice basis. One of its simple implementations is a recurring set of calls to SVP oracle solvers of dimension set by the block size parameter of the BKZ algorithm and LLL calls and will be used in the following for recovering the private key. Although these algorithms can be further refined to lead to better solutions for the SVP problem, the first vectors of the BKZ output are already short enough to give us candidates for lines of the private key matrix F.

In what follows we use the result of Proposition 2 to recover the private key matrix F in the following way.

Algorithm 1: Recovering private key F:

Let H the public key of the Matrix NTRU system with parameters n, p, q.

1. Compute $p^{-1} \mod q$and create the matrix L from eq.(3).
2. Run the BKZ algorithm on L and get L_{red}.
3. Use L_{red} to create a submatrix L^*_{red} with entries corresponding to the first n columns of L_{red}.
4. For each line f of the matrix F, check whether or not f is contained in one of the lines of L^*_{red}.

We implemented the above algorithm to assess the performance of the attack for several values of the parameters. Namely, we choose $q \in \{256, 4096\}$ for varying n and recorded what proportion of the lines of F one can recover from the attack using only the public key and public parameters p, q, n. We fix $p = 3$, a common choice in NTRU-like systems, as it simplifies modular arithmetic while preserving the essential algebraic structure needed for encryption and decryption. This setting is sufficient to investigate the system's behavior and assess the effectiveness of our attack strategies, since the other parameters are allowed to vary.

As showed in Figures 1 and 2, we see that the attack, in almost all experiments, can recover all lines of the matrix F for $n \in 40, 50, \ldots, 80$ and $q = 256$ and for $n \in 40, 50, \ldots, 110$ and $q = 4096$. In particular, breaking Matrix NTRU for $n = 110$ means breaking a private key that has $n^2 = 12100$ entries (which is equivalent to a huge private key polynomial in the classical NTRU), and the success of our experiments at this scale reveals a potential vulnerability.

For $q = 4096$, we also tested $n = 115$ and observed that the attack succeeded in approximately half of the experiments; however, in the other half, the BKZ algorithm stops the execution, indicating that the reduction process aborted before completion. This is expected, as the lattice dimension at this stage is $2n = 230$, which makes basis reduction more difficult. Although all the results in this work were obtained using a BKZ block size of 10, which already produced promising results, we increased the block size to investigate why BKZ stops the execution.

For $n = 115$, using the default BKZ block size of 10 or 15 caused the reduction to fail as we explained. However, increasing the block size to 20 or 25 resolved these issues, and in both cases the returned basis contained the private key, indicating a successful attack at the cost of increasing the running time (1 minute for block size 10 and 8 minutes for block size 25).

These results confirm that the attack remains feasible for higher dimensions, albeit at the cost of increasing the BKZ block size. Since the complexity of BKZ is exponential in the block size, the running time of the attack grows rapidly as the parameter increases [Chen *et al*., 2020b, Section 6.3].

One issue with the attack of algorithm 1 is that even though we can recover some lines of the matrix F, we still do not know how to reorder them to reconstruct the true F and use it for decryption. Nevertheless, we will see in the next Session that any permutation of lines of the private key F can be used to successfully decrypt a message encrypted with F and this allows us to construct a message recovery attack on the Matrix NTRU system.

5 Message recovery attack

In the previous section we showed how to construct the associated lattice for the Matrix NTRU system and demonstrated the viability of the attack for recovering, up to a permutation, the whole private key matrix F. In what follows we show that this attack can be used to successfully decrypt a message encrypted with F, even though the correct order of the lines of F is not known by the attacker.

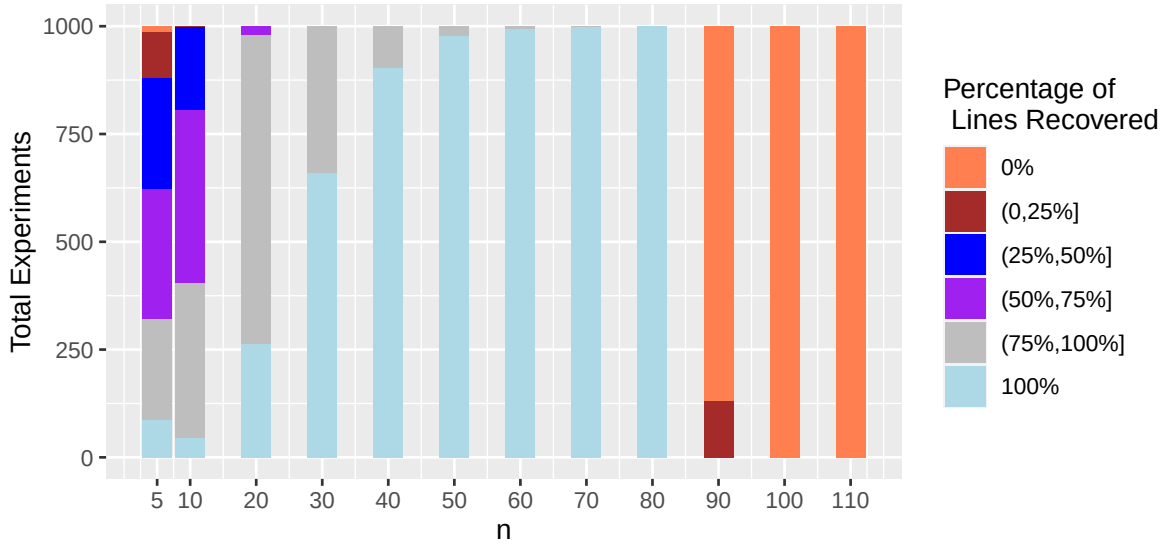
Proposition 3. *Let F, H be a corresponding private and public key pair for the Matrix NTRU system with parameters n, p, q and let E be a ciphertext associated with the encryption of a message M such that the decryption process of E using F correctly returns M. Let F^* be a matrix formed by permuting the lines of the matrix F. Then, applying the decryption process to E using F^* returns the message M.*

Proof. By the construction of F^*, there exists an uni-modular matrix D, such that $F^* = DF$. Applying the decryption process with F^* gives

$$A^* = F^*E \mod q = DF(HR + M) \mod q.$$

Using the definition of the matrix H and the fact that $FF_q = I$ we get,

Figure 1. Performance of the attack of algorithm 1 for recovering lines of the matrix F for the Matrix NTRU with parameters n and fixed $q = 256$. For each experiment, we generate new keys F and H and apply the attack recording how many experiments were able to recover all lines of F (100%), between 1 and $n - 1$ lines of F and none of the lines of F (0%).



$$\begin{aligned} A^* &= DF(HR + M) \mod q \\ &= DF(pF_qGR + M) \mod q \\ &= DpFF_qGR + DFM \mod q \\ &= pDGR + DFM \mod q \end{aligned} \tag{6}$$

The next decryption step is to compute A^* modulo p and notice that since D only change signs and permute lines of GR, we can still make this operation without incurring in the risk of extrapolating the norm of the entries of the matrix DGR. Therefore, we have

$$A^* \mod p = 0 + DFM = F^*M.$$

Finally, using the inverse F_p^* of $F^* \mod p$ we compute

$$C^* = F_p^*A^* = F_p^*F^*M \mod p = M.$$

□

We use the result of Proposition 3 in combination with Algorithm 1 to create a practical message recovery attack. This attack uses the upper left part of the reduced lattice basis returned by the BKZ algorithm as a potential key. As we saw in Section 4, this gives us in several cases the whole private key (up to a permutation of lines). The result of Proposition 3 says this potential key can still be used to decrypt a message successfully and this is tested in practice.

For each experiment, we generate new key pair F, H a new message M, encrypt it and try to decrypt with the key F^* returned from the attack of algorithm 1. The experiment was repeated 100 times on an Intel(R) Core(TM) i5-9500T CPU 2.20GHz and the success rate reported in Table 2. In the worst case scenario tested ($n = 110$ and $q = 4096$) we can recover the true message in less than one minute. For $q = 256$ the attack works for dimension until $n = 80$ and for $q = 4096$ it works for dimensions up to 100. Since Nayak *et al.* [2012] a Matrix NTRU of dimension n with an NTRU of dimension n^2, this clearly shows that Matrix NTRU is far weaker than NTRU since an NTRU with dimension $100^2 = 10\,000$ is far from being broken. In fact, the highest security suggested for NTRU in practical applications has parameters $n = 821$ and $q = 4096$ as shown in [Chen *et al.*, 2020a, Section 1.6]. By the results of this section, the Matrix NTRU of dimension n^2, should be comparable (in terms of the lattice attack) to at most an NTRU of dimension n. Therefore, even though the Matrix NTRU can encode a similar number of bits in the private key as the NTRU, its internal structure makes it vulnerable to attacks in a much smaller dimension. In addition, a direct consequence of Proposition 3 is that every private matrix F of dimension $n \times n$ has $n!$ equivalent keys, i.e., any of them can decrypt correctly. It is worth noticing that in the classical ntru there are n equivalent keys for each private key.

6 Applying May's Idea

As we can see in Figure 1, our attack against Matrix NTRU becomes ineffective as the lattice dimension increases. This is because the execution time and complexity of BKZ algorithm grow exponentially with the lattice dimension.

In 1999, Alexander May May [1999] presented a novel idea on the cryptanalysis of the NTRU cryptosystem. In his approach, he could work with lattices of smaller dimension compared to earlier attacks on NTRU. This was a key contribution because reducing the lattice dimension can make lattice reduction algorithms like LLL and BKZ more efficient.

Using lattice reduction and taking advantage of the special structure of NTRU secret keys, he was able to cut a certain number of columns in the matrix constructed over the NTRU lattice. These new lattices enabled him to break some medium security instances of NTRU, at that time, in less than 1 hour. Even for higher security levels, he could recover some keys, namely the weak ones.

Figure 2. Same settings as in Figure 1 but with $q = 4096$.

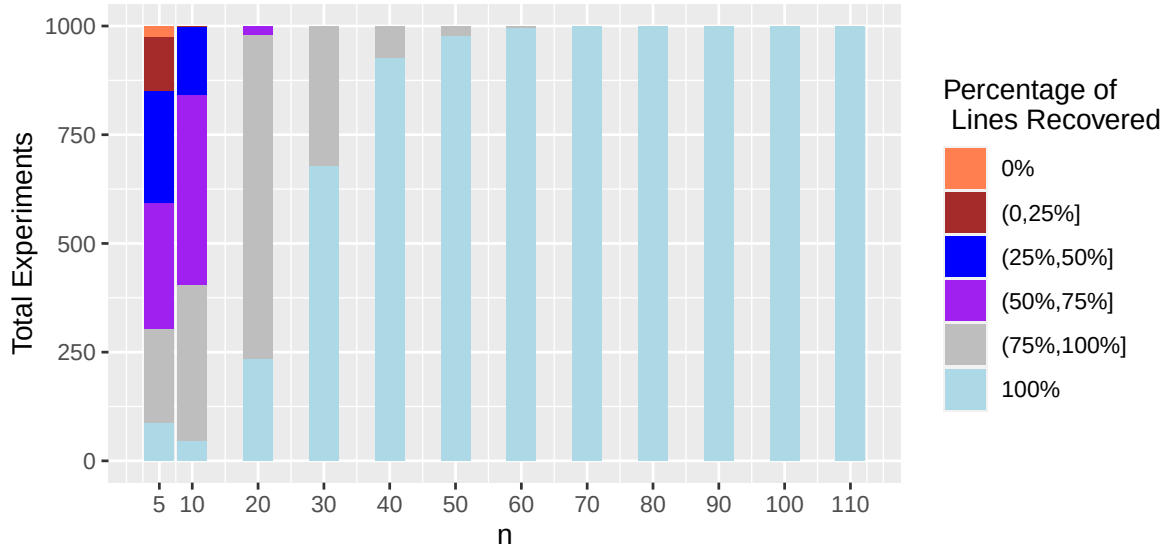


Table 2. Performance of the message recovery attack for the Matrix NTRU with parameters n and fixed $q \in \{256, 4096\}$. For each experiment, we generate new key pair F, H a new message M, encrypt it and try to decrypt with the key F^* returned from the attack of algorithm 1. The experiment was repeated 100 times and the success percentage recorded.

q	$n = 5$	$n = 10$	$n = 20$	$n = 30$	$n = 40$	$n = 50$	$n = 60$	$n = 70$	$n = 80$	$n = 90$	$n = 110$
256	1	1	1	1	1	1	1	1	0	0	0
4096	1	1	1	1	1	1	1	1	1	1	1

As we can see in Silva *et al*. [2024], we can apply May´s technique to recover keys over NTRU variants. That's the case in this section, where we are going to apply May's idea to improve the results of the previous sections.

The idea is quite simple. Instead of using all possible basis vectors, we can select only those most relevant to find short vectors and, thus, calculate the private key on Matrix NTRU cryptosystem. This reduces the computational complexity of lattice reduction algorithms, which enables us to apply BKZ algorithm for larger matrix dimensions.

In this work, as in Silva *et al*. [2024], we will reduce lattice dimension by removing columns, even though this procedure can introduce some inaccuracies in the lattice structure. However, despite potential inaccuracies, the reduced lattice may retain enough information to enable private key recovery.

Columns can be removed randomly, but the attack's effectiveness depends on which columns are discarded. May's strategy aims to lower the lattice dimension while preserving vectors that contain critical information. If the wrong columns are removed, the reduced lattice may lack the necessary properties for a successful attack. In this work, column removal was performed on the right side of the matrix.

Therefore, even after reducing the original lattice dimension, we can recover the private key in the Matrix NTRU cryptosystem. Next, we are going to see the experimental results.

6.1 Experimental results of the lattice attack based on Mays idea

Figure 3 summarizes the results we obtain through the application of May's idea to recover the private key of the Matrix NTRU system. Using the new lattice with smaller dimension we could attack the Matrix NTRU system for dimensions up to 120, which was not possible on a personal computer using out of the box BKZ as was done in Section 4.

To appreciate the impact of May's idea, recall that in Section 4 we applied BKZ to a lattice of dimension $2n$. The attack was successful when we were able to recover the entire private key, that is, when the reduced lattice contained all n rows of the private key. The practical limitation of the attack lies in the required computational resources, which constrained our experiments to $n = 110$ (a setting where both the attack and BKZ consistently succeeded on a personal computer). May's approach offers two key advantages:

1. It extends the range of parameters for which the attack is effective (since for a fixed n, the attack can be executed by applying BKZ to a lattice of smaller dimension $2n-k$ for $k > 0$);
2. Running BKZ on a lattice of smaller dimension is faster.

The experiment was designed to recover all lines of the private key of the Matrix NTRU system for fixed $p = 3$, $n \in \{110, 115, 120, 125\}$ and a dimension cut $k \in \{0, 1, \ldots, n-1\}$. For each n we generate 100 different public private key pairs and attack the lattice to recover the private key. The upper panel of **Figure 3** reports on the success rate of the attack and the lower panel reports on whether BKZ runs successfully. These results show that using May's idea still gives

Table 3. Success rates for different strategies of column removal when using May's technique.

n	Strategy	$k = 30$	$k = 40$	$k = 50$	$k = 60$	$k = 70$
100	Last Columns	100.00%	99.00%	97.02%	89.11%	0.00%
100	Random Columns	100.00%	99.00%	100.00%	91.09%	0.00%
110	Last Columns	100.00%	95.05%	84.16%	59.41%	0.00%
110	Random Columns	100.00%	94.06%	90.10%	62.38%	0.00%
115	Last Columns	67.33%	51.49%	39.60%	7.92%	0.99%
115	Random Columns	65.35%	57.43%	39.60%	11.88%	0.99%

an attack that works with success rate above 50% even for cut values up to 60.

For $n = 115$, the success rate starts at around 50% when we use the full lattice ($k = 0$). As we increase the value of k, the lower panel of Figure 3 shows that BKZ runs successfully more often and the attack success rate increases up around 80% (k = 17). As k continues to increase, the success rate starts decreasing reaching 0% for a cut $k \geq 71$.

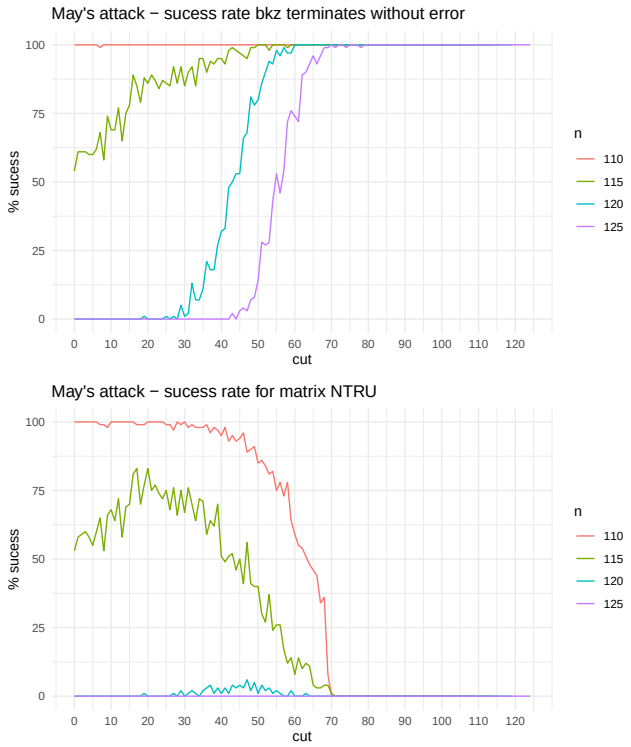
For $n = 120$ the success rate starts at zero due to the fact that BKZ does not run successfully for low values of k as the lattice dimension $240 - k$ is quite high (**Figure 3**, lower panel). As discussed in the previous section, one could increase the block size in BKZ to avoid these errors at the cost of increasing the computational time. For these experiments, we stayed in the fixed scenario where the block size is only 10 as this suffices to show the effectiveness of using May's idea.

Still for $n = 120$, and as the cut value k increases, $k = 19$, one instance out of 100 runs successfully and it indeed contains the private key. As k continues to increase, BKZ starts to run successfully more often and the success rate of the attack increases up to 6% for $k = 47$ and then starts to decrease to zero. For $k \geq 64$, although BKZ runs successfully, the attack does not work as we are cutting too many dimensions of the original lattice and, therefore, we will not be able to recover the private key.

The case $n = 125$ did not return the whole private key and therefore its success rate was zero. Recall that for our attack to work, we need to solve SVP n times to find n lines of the private key matrix. Although in some cases it found a couple of lines of the private key matrix, it did not succeed in finding all lines of the matrix.

One could ask if the way we selected the columns to cut can affect the attack performance significantly. We have already observed at the introduction of Section 6 that the attack performance cannot be significantly improved by changing the strategy on how the columns are removed and this is backed up by an experiment whose results are shown in Table 3. We selected different lattice dimensions and 5 cut values k and repeated the experiment 100 times for each combination of values. Out of the 15 difference scenarios, removing columns at random was slightly better in 6 cases, and its performance was exactly the same at the other scenarios or worse (2 of them).

Figure 3. Lattice attack to recover all lines of the private key of the Matrix NTRU system for fixed $p = 3, q = 4096, n \in \{110, 115, 120, 125\}$ and a dimension **cut** $k \in \{0, 1, \ldots, n-1\}$. For each n we generate 100 different public private key pair and attack the lattice to recover the private key. The upper panel reports on the success rate of the attack and the lower panel reports on whether BKZ runs successfully.



7 Conclusion

In Nayak *et al*. [2012], the authors compare the speed performance for encryption and decryption of the classical NTRU and the Matrix NTRU arguing that Matrix NTRU is faster than NTRU for comparable parameters. They compare a Matrix NTRU with parameters n with an NTRU with parameter n^2, since the original idea of the Matrix NTRU system is to encode the private key polynomial of degree n^2 into a matrix of dimension $n \times n$. In terms of brute force search for the private key they are comparable. In terms of lattice-based attacks, the post-quantum NIST finalist NTRU encrypt (Chen *et al*. [2020a]) with parameters $n = 509$ and $q = 2048$ has already moderate security and cannot be broken by current the available techniques using lattice attacks. The equivalent Matrix NTRU would have dimension around 23, an integer approximation to $\sqrt{509}$. However, a Matrix NTRU with parameters $n = 23$ is completely vulnerable to lattice attacks on a personal computer as showed in Section 4. In fact we can attack such system for even higher dimension.

These results also show that in terms of lattice security, selecting parameters for using Matrix NTRU for a key exchange protocol would require using at least $n = 509$ for a similar security to NTRU encrypt (level 1 security according to NIST specification in 2020). However, it does not make sense to exchange a private key with n^2 entries when there are more efficient post-quantum key exchange systems available for the same security level.

The bottleneck of the proposed attack is in finding a suitable short vector for the Matrix NTRU system using lattice basis reduction algorithms. Therefore, any improvement on these techniques such as replacing the BKZ routine by, e.g., the ones described in (Albrecht and Ducas [2021], May and Silverman [2001], and Bi and Han [2021]; Zhao and Ye [2023]) would improve the results showed in **Figures 1 and 2**.

The lattice attack is inherent to NTRU-type systems and forms the basis for evaluating their security against classical attacks. In our approach, the success of the attack stemmed from exploiting the structural weakness of the private key by targeting its rows independently. A potential countermeasure could involve designing a new variant of matrix-NTRU that incorporates both left and right matrix multiplications during public key generation and encryption/decryption. Such modifications could disrupt the independence of private key components across rows, thereby increasing the diffusion of cryptographic quantities and making attacks more difficult. This direction presents an interesting avenue for future research. However, it is worth noting that matrix multiplication tends to offer inherently lower diffusion compared to polynomial convolution (used e.g. in Bernstein *et al*. [2017]; Tripathi *et al*. [2016]), which may itself pose a security concern.

The vulnerability of the Matrix NTRU system is significant, as the presented lattice attacks effectively reduce its security from n^2 to n. Comparing with the NTRU cryptosystem, the Matrix NTRU tries to create a faster variant of NTRU by separating the private key into slices (lines of a matrix) and using this to create the public key. On the other hand, this come at a very high security cost since just one line is used at a time to construct the lines of the public key. That means it runs faster at the cost of reducing the diffusion during the creation of the public key. On the other hand, NTRU creates public keys by using convolution of polynomials, and therefore, every coefficient contributes to create every coefficient of the public key. In addition, we have increased the attack performance by showing how May's approach of lattice dimension reduction can be applied successfully in the Matrix NTRU system. We show that it is possible to extend the applicability of BKZ algorithm to this system by using a similar approach to that used to attack NTRU.

In this work, we showed that it is possible to decrypt ciphertexts in Matrix NTRU cryptosystem if we just have a matrix whose columns are a permutation of the original columns of the private key. As a future work, inspired by Coppersmith and Shamir [1997], one could investigate the viability of decryption using matrices with small norm, i.e., ternary matrices with approximately the same number of non-zero elements as the private key.

To conclude, we believe that the proposed attack can be adapted to other NTRU like systems relying on matrices operations such as the one based on the Gaussian integers.

Declarations

Authors' Contributions

All authors contributed equally to the conception, design, execution, analysis, and writing of the manuscript. All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Acknowledgements

The authors thank anonymous reviewers for their valuable and constructive comments, which helped to improve the quality of this work.

Availability of data and materials

The Matrix NTRU system and the attack implementations generated during this study are publicly available at `https://github.com/thiagodoregosousa/matrix-ntru.git`

References

Albrecht, M. and Ducas, L. (2021). Lattice attacks on ntru and lwe: a history of refinements. *Cryptology ePrint Archive*. Available at:`https://eprint.iacr.org/2021/799`.

Bernstein, D. J., Chuengsatiansup, C., Lange, T., and van Vredendaal, C. (2017). Ntru prime: Reducing attack surface at low cost. In *Selected Areas in Cryptography (SAC 2017)*, pages 235–260. Springer. DOI: 10.1007/978-3-319-72565-9$_1$2.

Bi, J. and Han, L. (2021). Lattice attacks on ntru revisited. *IEEE Access*, 9:66218–66222. DOI: 10.1109/ACCESS.2021.3076598.

Bremner, M. (2011). *Lattice basis reduction*. CRC Press New York. DOI: 10.1201/b11066.

Chen, C., Danba, O., Hoffstein, J., Hulsing, A., Rijneveld, J., Schanck, J. M., Schwabe, P., Whyte, W., and Zhang, Z. (2020a). Ntru: algorithm specifications and supporting documentation (2019). 1. Available at:`https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions`.

Chen, C., Danba, O., Hoffstein, J., Hülsing, A., Rijneveld, J., Schanck, J. M., Saito, T., Schwabe, P., Whyte, W., Xagawa, K., Yamakawa, T., and Zhang, Z. (2020b). NTRU algorithm specifications and supporting documentation. Section 6.3. DOI: 10.1007/978-1-4419-5906-5$_4$64.

Chen, Y. and Nguyen, P. Q. (2011). Bkz 2.0: Better lattice security estimates. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 1–20. Springer. DOI: 10.1007/978-3-642-25385-0_1.

Coglianese, M. and Goi, B.-M. (2005). Matru: A new ntru-based cryptosystem. In *Progress in Cryptology-INDOCRYPT 2005: 6th International Conference on*

Cryptology in India, Bangalore, India, December 10-12, 2005. Proceedings 6, pages 232–243. Springer. DOI: 10.1007/11596219_19.

Coppersmith, D. and Shamir, A. (1997). Lattice attacks on ntru. In *International conference on the theory and applications of cryptographic techniques*, pages 52–61. Springer. DOI: doi.org/10.1007/3-540-69053-0_5.

do Rêgo Sousa, T. and Neto, T. S. (2025). Improved decryption bounds and key generation for matrix ntru over integral domain. In *Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg)*, pages 515–528. SBC. DOI: doi.org/10.5753/sbseg.2025.9721.

Gama, N. and Nguyen, P. Q. (2007). New chosen-ciphertext attacks on ntru. In *International Workshop on Public Key Cryptography*, pages 89–106. Springer. DOI: 10.1007/978-3-540-71677-8_7.

Hall, C., Goldberg, I., and Schneier, B. (1999). Reaction attacks against several public-key cryptosystem. In *Information and Communication Security: Second International Conference, ICICS'99, Sydney, Australia, November 9-11, 1999. Proceedings 2*, pages 2–12. Springer. DOI: 10.1007/978-3-540-47942-0_2.

Hoffstein, J., Pipher, J., and Silverman, J. H. (1998). Ntru: A ring-based public key cryptosystem. In *International algorithmic number theory symposium*, pages 267–288. Springer. DOI: 10.1007/bfb0054868.

Howgrave-Graham, N., Nguyen, P. Q., Pointcheval, D., Proos, J., Silverman, J. H., Singer, A., and Whyte, W. (2003). The impact of decryption failures on the security of ntru encryption. In *Annual International Cryptology Conference*, pages 226–246. Springer. DOI: 10.1007/978-3-540-45146-4_14.

Jacques-García, F. A., Uribe-Mejía, D., Macías-Bobadilla, G., and Chaparro-Sánchez, R. (2022). On modular inverse matrices a computation approach. *South Florida Journal of Development*, 3(3):3100–3111. DOI: 10.46932/sfjdv3n3-005.

Jaulmes, É. and Joux, A. (2000). A chosen-ciphertext attack against ntru. In *Annual international cryptology conference*, pages 20–35. Springer. DOI: 10.1007/3-540-44598-6_2.

Kumar, V., Mamdikar, M. R., and Gosh, D. (2013). Matrix formulation of ntru algorithm using multiple public keys from matrix data bank for high degree polynomials. In *CEEE*, pages 191–198. DOI: 10.15224/978-981-07-6260-5-40.

Lenstra, A. K., Lenstra, H. W., and Lovász, L. (1982). Factoring polynomials with rational coefficients. *Mathematische annalen*, 261:515–534. DOI: 10.1007/BF01457454.

Luo, X.-R. and Lin, C.-H. J. (2011). Discussion on matrix ntru. *IJCSNS International Journal of Computer Science and Network Security*, 11(1):32–35. Available at:`http://paper.ijcsns.org/07_book/html/201101/201101004.html`.

Mamdikar, M. R., Kumar, V., and Ghosh, D. (2013). Implementation of automatic invertible matrix mechanism in ntru matrix formulation algorithm. Available at:`https://irdindia.in/journal_ijacte/pdf/vol2_iss3/7.pdf`. Accessed: 2026-04-17.

May, A. (1999). Cryptanalysis of ntru. *preprint, February*. Available at:`https://www.scirp.org/reference/referencespapers?referenceid=653326`.

May, A. and Silverman, J. H. (2001). Dimension reduction methods for convolution modular lattices. In *International Cryptography and Lattices Conference*, pages 110–125. Springer. DOI: 10.1007/3-540-44670-2_10.

Mittal, S. and Ramkumar, K. (2022). A retrospective study on ntru cryptosystem. In *AIP Conference Proceedings*, volume 2451. AIP Publishing. DOI: 10.1063/5.0095312.

Nayak, R., Pradhan, J., and Sastry, C. (2011). Reaction attacks in the matrix scheme of ntru cryptosystem. In *International Conference on Advances in Information Technology and Mobile Communication*, pages 27–32. Springer. DOI: 10.1007/978-3-642-20573-6_5.

Nayak, R., Pradhan, J., and Sastry, C. V. (2012). Evaluation of performance characteristics of polynomial based and lattice based ntru cryptosystem. *International Journal of Network Security*. Available at:`https://www.academia.edu/33351040/Evaluation_of_Performance_Characteristics_of_Polynomial_based_and_Lattice_based_NRTU_Cryptosystem`.

Nayak, R., Sastry, C., and Pradhan, J. (2008). A matrix formulation for ntru cryptosystem. In *2008 16th IEEE International Conference on Networks*, pages 1–5. IEEE. DOI: 10.1109/icon.2008.4772602.

Salleh, N. and Kamarulhaili, H. (2020). Ntru public-key cryptosystem and its variants: An overview. *Int. l J. of Cryptology Research*, 10(1):1–21. Avaialble at:`https://mscr.org.my/data/journal/journal-20200507123724.pdf`.

Silva, A., Sousa, T., and Neto, T. S. (2024). Cutting dimensions in the lll attack for the etru post-quantum cryptosystem. In *Anais do XXIV Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, pages 154–164, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/sbseg.2024.240859.

Silverman, J. H., Pipher, J., and Hoffstein, J. (2008). *An introduction to mathematical cryptography*, volume 1. Springer. DOI: /10.1080/01611190902721016.

Singh, S. and Padhye, S. (2016). Generalisations of ntru cryptosystem. *Security and Communication Networks*, 9(18):6315–6334. DOI: 10.1002/sec.1693.

Team, S. D. (2024). *SageMath*. Available from `https://www.sagemath.org`.

Tripathi, B., Thakur, K., Nayak, R., Sastry, C., and Pradhan, J. (2016). Ntru cryptosystem with companion matrix. *A matrix formulation for NTRU cryptosystem*, pages 1–5. Available at:`https://www.semanticscholar.org/paper/NTRU-Cryptosystem-with-Companion-Matrix-Tripathi-Thakur/4c21dddc146c077db2f709d21967a34f8ed3a3a3`.

Wijayanti, I. E., Isnaini, U., Sari, A. K., Ali, S., and Aji, N. C. (2023a). Matrix ntru cryptosystem over integral domain. In *2023 IEEE International Conference on Cryptography, Informatics, and Cybersecurity (ICoCICs)*, pages 35–40. IEEE. DOI: 10.1109/ICoCICs58778.2023.10277330.

Wijayanti, I. E. *et al*. (2023b). The meet-in-the-middle attack on the matrix ntru cryptosystem. In *2023 IEEE*

International Conference on Cryptography, Informatics, and Cybersecurity (ICoCICs), pages 149–153. IEEE. DOI: 10.5753/sbseg.2024.240851.

Zhao, Z. and Ye, Q. (2023). Revisiting lower dimension lattice attacks on ntru. *Information Processing Letters*, 181:106353. DOI: 10.1016/j.ipl.2022.106353.

Optimizing ML-KEM with OpenMP: A Multilevel Parallelization Study

Vinícius Lagrota[1], **Tertuliano Souza Neto**[1],
Thiago do Rêgo Sousa[1], **Fábio Maia**[2]

[1] Research and Development Center for Communication Security (CEPESC)
Brasília, DF – Brazil.

[2]Integrated Center for Advanced Security Systems (CISSA)
Recife, PE – Brazil.

vinicius.lagrota@gmail.com, tsouzaneto@gmail.com,
thiagodoregosousa@gmail.com, fmww@cesar.org.br

Abstract. *In this work, we present* `libcesarpq`*, a cryptographic library that integrates an optimized implementation of the post-quantum Key Encapsulation Mechanism ML-KEM. We perform a detailed analysis of the algorithm's source code to identify opportunities for parallelization using OpenMP directives. We show that, since most core ML-KEM functions have very short execution time, the overhead introduced can severely limit performance gain. Therefore, we evaluate different parallelization strategies at both the core-layer functions and the API-layer, as well as the batched API-layer. The results are compared with the widely used post-quantum library* `liboqs` *in both single-threaded and multithreaded environments, showing comparable performance on the* `x86-64` *platform and a performance advantage for* `libcesarpq` *on the optimized* `aarch64` *platform.*

1. Introduction

Three decades ago, Peter Shor introduced a quantum algorithm capable of breaking several widely used public-key cryptosystems [Shor 1994]. Shortly thereafter, Lov Grover introduced a quantum search algorithm [Grover 1996] that can potentially reduce the effective security level of symmetric-key cryptosystems by a factor of two. The combined impact of Shor's algorithm on public-key schemes and Grover's algorithm on symmetric primitives motivated the cryptographic community to develop what is now known as post-quantum cryptography (PQC). Consequently, the design of robust cryptographic schemes based on hard mathematical problems that remain intractable for both classical and quantum computers has become an active and rapidly growing research area. Indeed, research and development in PQC have significantly accelerated over the past decade.

In this context, the National Institute for Standards and Technology (NIST) launched a public PQC standardization project [Chen et al. 2016] with the goal of selecting and standardizing public-key cryptographic algorithms for key encapsulation, shared secret establishment, and digital signatures. After four evaluation rounds spanning about six years, NIST selected one algorithm for key encapsulation mechanism (KEM) and three algorithms for digital signatures. The chosen KEM algorithm, originally known

as CRYSTALS-Kyber, was standardized as *Module-Lattice-Based Key-Encapsulation Mechanism Standard* (ML-KEM) in the Federal Information Processing Standards Publication (FIPS)-203 specification [NIST 2024].

The security of ML-KEM algorithm relies on the hardness of the Module-Learning With Errors (MLWE) problem, a generalization of the Learning With Errors (LWE) problem introduced by Regev [Regev 2005] and formally studied in the context of modules by Langlois and Stehlé [Langlois and Stehlé 2015]. ML-KEM belongs to the family of lattice-based cryptosystems and supports three security categories: ML-KEM-512, providing security comparable to AES-128 (NIST security category 1); ML-KEM-768, with security comparable to AES-192 (NIST security category 3); and ML-KEM-1024, offering security comparable to AES-256 (NIST security category 5).

These three security categories are closely related to the parameter k, which determines the module dimension used in the underlying MLWE problem. In particular, the scheme operates over the k-fold cartesian product of the polynomial ring R_q, denoted by R_q^k, where $R_q = \frac{\mathbb{Z}_q[X]}{(X^n+1)}$. In practice, each security category corresponds to a specific value of k. More precisely, ML-KEM-512 uses $k = 2$, ML-KEM-768 uses $k = 3$, and ML-KEM-1024 uses $k = 4$. Increasing the value of k enlarges the dimension of the module lattice and consequently increases the computational hardness of the underlying MLWE problem. As a result, higher values of k provide stronger security guarantees, at the cost of larger key sizes and higher computational complexity.

By combining strong theoretical foundations with practical efficiency [NIST 2024, Avanzi et al. 2019, Bos et al. 2018], ML-KEM represents a cornerstone of the PQC transition. However, despite its suitability for standardization, the computational cost of post-quantum schemes can still pose challenges, particularly in resource-constrained environments [Kim and Seo 2025, Lagrota et al. 2025] or in settings where the algorithm must be executed in batches. This motivates the exploration of optimization techniques, such as parallel programming, to further improve performance.

1.1. Related Works

Improving the performance of ML-KEM and related post-quantum schemes has attracted significant attention due to the high computational cost of lattice-based cryptography. One line of work focuses on improving the software implementation itself. [Paiva et al. 2025] proposed tailored error-correction codes for lattice-based KEMs and implemented compact ML-KEM instantiations with negligible performance impact. [Becker et al. 2022] accelerated lattice-based schemes on Armv8-A through optimized Number Theoretic Transform (NTT) arithmetic and Neon vectorization, while [Wei et al. 2026] presented an optimized ML-KEM implementation on ARMv9-A using SVE2 and SME. These works show that software- and code-level optimizations can substantially improve performance.

Another line of research explores hardware-assisted acceleration. [Gewehr et al. 2024] investigated instruction-set extensions for Kyber/ML-KEM on RISC-V embedded systems, while [Ni et al. 2025] and [Nguyen et al. 2025] proposed efficient hardware architectures for ML-KEM. In addition, [Lagrota et al. 2022] presented a configurable hardware/software co-design combining an ARM processor with FPGA acceleration. Overall, these studies demonstrate the benefits of specialized hardware support, custom instruction sets, and hardware/software co-design.

A third direction investigates parallel execution. [Azevedo et al. 2025] proposed selective parallelization of ML-KEM on a dual-core ESP32, showing that the gains depend strongly on function granularity and parallelization overhead. However, the literature still lacks a systematic study of OpenMP-based parallelization for ML-KEM on general-purpose multicore processors. In particular, there is still limited understanding of how OpenMP behaves across different abstraction levels in PQC, especially in ML-KEM, ranging from fine-grained internal routines to higher-level, more computationally demanding operations. This work addresses that gap.

1.2. Our Contributions

The main contributions of this work can be summarized as follows:

- A detailed analysis of the ML-KEM algorithm, identifying opportunities for parallelization and investigating OpenMP-based strategies at different abstraction levels of the implementation.
- An extensive experimental evaluation across x86-64 and AArch64 architectures, assessing the impact of parallelization at the core-layer, the Application Programming Interface (API)-layer, and the batched API-layer functions.
- The design and implementation of the post-quantum cryptographic library `libcesarpq`, which incorporates the proposed OpenMP-based parallelization strategies and serves as the experimental platform for the analyses presented in this work.

2. Preliminaries

A dramatic shift in processor design has changed the history of microprocessor performance. Although the last quarter of the 20th century saw an unprecedented growth in processor clock speeds, the current century has seen a marked slowdown in this trend [Pacheco and Malensek 2021]. As a result, contemporary performance gains are driven primarily by parallelism rather than by increases in single-processor speed. In response to this trend, the industry shifted its focus toward the integration of multicore processors onto a single chip. Such processors consist of multiple single-core units and taking advantage of their capabilities requires the development of software that exploits the power of multiple processing elements, commonly referred to as parallel programs. Therefore, parallel programming plays a central role in a wide range of application domains, including climate modeling, drug discovery, energy research, data analysis, and cryptography.

As parallel computing developed, the need to measure the performance of parallel programs emerged, which is accomplished through performance metrics. Beyond simply measuring execution time, these metrics make it possible to identify potential bottlenecks in code execution and to determine whether parallel programming should, in fact, be employed for a particular application, since in some cases performance may even be worse than that of a serial execution. The goal of executing a procedure in parallel is to accelerate the execution of a given algorithm. At least three factors affect processing time: the type of hardware used, the degree of parallelism inherent to the algorithm, and the parallelization model adopted.

The metric that quantifies the benefit of parallelizing an algorithm is known as *speedup*. Let T_s denote the execution time of the algorithm in its serial form, that is,

without parallelism, and let T_p denote the execution time under parallel execution with p threads. Speedup is then defined as

$$S(p) = \frac{T_s}{T_p}. \tag{1}$$

In the ideal scenario, the entire computational workload is evenly distributed among the p threads, with no overhead due to thread creation, synchronization, communication, scheduling, or idleness. In this case, the speedup is given by $S(p) = p$. When this condition is achieved, the parallel program is said to exhibit *linear speedup*.

A useful way to express the trade-off between the execution time of the algorithm in its serial form (i.e., useful work), T_s, and overhead introduced by parallel execution with p threads, $T_{oh}(p)$, is to model the total execution time as

$$T_p(p) = \frac{T_s}{p} + T_{oh}(p), \tag{2}$$

where $T_p(p)$ denotes the total parallel execution time measured when using p threads. In practical implementations, the overhead $T_{oh}(p)$ may include thread-management costs, synchronization, barriers, scheduling, and other costs not associated with useful computation. Equation 2 highlights that increasing the number of threads reduces the time consumed by the ideally parallelizable portion of the computation, represented by T_s/p, but does not eliminate the overhead term. On the contrary, depending on the implementation and the execution environment, $T_{oh}(p)$ may remain significant or even increase with the number of threads. Therefore, although parallel execution may reduce the total execution time, the resulting speedup is typically sublinear.

3. Implementation Strategy for Parallel ML-KEM using OpenMP

Since many cryptographic algorithms and libraries are implemented in the C language and are typically executed on general-purpose processing units (CPUs), OpenMP is a suitable choice for optimizing such functions due to its simplicity and high portability, as it is supported by multiple compilers, architectures, and operating systems. In this context, this section details the API extension and the internal modifications introduced to incorporate OpenMP into ML-KEM. Our analysis considers three abstraction layers in the ML-KEM implementation, as illustrated in Figure 1.

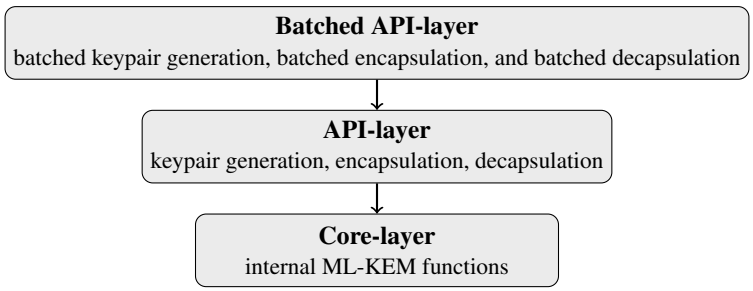


Figure 1. Three abstraction layers in the ML-KEM analysis.

The lowest abstraction layer — the core-layer — is composed of the internal ML-KEM functions, which fundamentally depend on the ML-KEM security level (k,

defined in Section 1). Above it, the API-layer includes the functions provided by the original ML-KEM API — keypair generation, encapsulation, and decapsulation — and relies on the core-layer functions. Finally, the uppermost layer is the batched API-layer. This layer extends the original API by allowing the API-layer functions to be processed in batches — namely, batched keypair generation, batched encapsulation, and batched decapsulation.

To support this analysis while preserving the correctness of the cryptographic operations, we developed a cryptographic library called `libcesarpq`[1], which supports both single-threaded and multithreaded operations depending on compilation and configuration parameters. The library automatically detects the number of available cores, thereby improving usability, but it also provides an API through which users can limit the maximum number of threads. This cryptographic library allows the functions used in the ML-KEM algorithm to be executed with the appropriate OpenMP directives when enabled on different architectures. We also adopted an incremental development strategy: starting from a correct and fully functional baseline implementation, we progressively introduced parallel constructs and verified correctness after each modification.

Sections 3.1 through 3.3 discuss each abstraction layer shown in Figure 1, ranging from the lowest layer (core-layer) to the highest layer (batched API-layer).

3.1. Core-layer

This section discusses how OpenMP was included in the core-layer — i.e., applied to the internal ML-KEM functions — using the `mlkem-native`[2] implementation as a starting point. The overall strategy adopted to optimize the internal ML-KEM functions was to analyze the source code of each internal function involved in the three main operations (keypair generation, encapsulation, and decapsulation), together with their respective subroutines, and identify iterative blocks that account for a significant portion of the total computation time and can be executed independently. These blocks were then selected as candidates for parallel regions executed by independent threads.

A preliminary analysis of the individual internal ML-KEM functions revealed that the true potential for parallelization is in the functions related to polynomial algebra. The existence of `for` loops in most of the functions make them good initial candidates for the inclusion of parallel regions. Furthermore, it was verified that the `for` loops are independent, because each loop operates in a polynomial ring of the ML-KEM algorithm, which are independent from each other.

Most of the functions have loops of size varying according to the security level $k \in \{2, 3, 4\}$. This is the case for functions such as `mlk_polyvec_{mulcache_compute, tobytes, frombytes, add, compress_du, ntt, invntt_tomont, tomont, assert_bound_2d}`, `mlk_polymat_permute_bitrev_to_custom`, and `mlk_matvec_mul`. Therefore, we have added parallel regions in these loops by inserting the directive `#pragma omp parallel`. The same strategy was used for the functions `mlk_polyvec_assert_bound_2d` and

[1]More specific, in this work, we used `libcesarpq-v1.5.1`, available in `https://gitlabcissa.cesar.org.br/cepesc/libcesarpq/-/releases/1.5.1`.

[2]It is a secure, fast, and portable implementation of ML-KEM. Available at `https://github.com/pq-code-package/mlkem-native` and also deployed in the widely known `liboqs` library.

`mlk_polyvec_basemul_acc_montgomery_cached_c` whose loop size is $N/2 = 128$, where $N = 256$ for all security levels, and the `mlk_gen_matrix` function whose loop size is $\texttt{nblocks} = (k * k)/4 \in \{1, 2, 4\}$.

As an illustrative example, we analyze the internal ML-KEM function `mlk_gen_matrix`, shown in Code 1. To evaluate the speedup achieved with OpenMP directives, parallel constructs were applied where beneficial, and the same strategy was extended to the remaining internal functions.

Code 1. Snippet of `mlk_gen_matrix`.

```
1  void mlk_gen_matrix(mlk_polymat *a, const uint8_t seed, ... )
2  {
3      const unsigned nblocks = (MLKEM_K * MLKEM_K / 4);
4      ...
5      #pragma omp parallel for ... num_threads(omp_get_max_threads
           () < (int)nblocks ? omp_get_max_threads() : (int)nblocks)
6      for (unsigned b = 0; b < nblocks; b++)
7      {
8          // Operations executed in parallel involving polynomial
               and seed
9          ...
10     }
11     //Remaining operations executed serialized
12     ...
13 }
```

The operations performed in the `for` loop of the function `mlk_gen_matrix`, described in Code 1, are processed parallelized, independently for each value of the variable `b`, which is incremented within the loop. This independence is the primary reason for executing this loop in parallel.

Another factor affecting OpenMP speedup is thread-management overhead (see Equation 2). As a first countermeasure, in all parallel sections we bounded the number of threads by the maximum number of physical cores available on the target hardware. Initial benchmarking further showed that increasing the number of threads beyond the number of loop iterations did not yield any performance improvement. In this situation, the additional threads do not decrease the sequential time of useful parallel work ($\frac{T_s}{p}$ in Equation 2), but only add runtime overhead related to thread coordination, iteration scheduling, and synchronization (T_{oh} in Equation 2). To reduce this overhead, we limited the number of threads according to both the loop size and the number of available cores.

3.2. API-layer

The API-layer implements the original ML-KEM interface, including the conventional keypair generation, encapsulation, and decapsulation functions. No OpenMP-based parallelization is implemented directly at this layer. However, it relies on the core-layer, discussed previously in Section 3.1, where OpenMP parallelization is employed. Therefore, the API-layer makes indirect use of OpenMP through its dependence on the core-layer.

3.3. Batched API-layer

Certain high-performance scenarios would be benefited by batched execution of key exchange operations. Two representative examples are high-throughput Transport Layer Security (TLS) servers, which may need to process thousands of handshakes per second, and secure group messaging systems, in which a single sender establishes shared secrets with multiple recipients simultaneously.

To address these scenarios, the `libcesarpq` library implements an extended API for batched ML-KEM operations, namely batched API-layer. This extension provides wrapper functions that iterate over a fixed number of operations — informed by the user — and execute them in parallel using OpenMP. In particular, the extended API exposes the functions batched keypair generation, batched encapsulation, and batched decapsulation. These functions use OpenMP directives to distribute the workload across threads, following a strategy similar to that described in Section 3.1, but applied at the batch level through the API-layer.

In the batched setting, however, each operation invokes the single-threaded version of the underlying ML-KEM implementation; thus, nested parallelism is intentionally avoided. Furthermore, the batched functions use OpenMP's `schedule(static)` directive, since the number of operations is known in advance, informed by the user in the function arguments, which helps reduce scheduling overhead. They also limit the number of threads according to both the number of batch operations and the number of available cores, thereby avoiding unnecessary thread-management overhead (T_{oh} in Equation 2).

4. Experimental Results for Parallel ML-KEM

Building on Section 3, this section presents the experimental results for the parallel ML-KEM implementation using OpenMP. Among the three NIST security levels for ML-KEM, this work focuses on the lowest security level (ML-KEM-512) and highest security level (ML-KEM-1024) allowing to highlight the difference between them. We evaluate two platforms: an Intel(R) Xeon(R) 6710E in a virtualized environment with 12 available cores and no acceleration instruction set enabled (e.g., SSE2, SSE4, AVX2)[3], denoted `x86-64-ref`; and an Apple Silicon M2 Pro operating natively with 10 available cores (6 performance and 4 efficiency) and all accelerations enabled, denoted `aarch64-native`. These contrasting environments were intentionally chosen to assess the impact of OpenMP under different execution conditions.

For the API-layer and batched API-layer functions, we also compare `libcesarpq` with `liboqs`, an open-source C library widely used as a benchmarking baseline for PQC algorithms [Abbasi et al. 2025]. Both libraries were compiled with the most suitable settings for each target architecture. Section 4.1 analyzes the internal ML-KEM functions, Section 4.2 discusses the API-layer, and Section 4.3 presents the batched API-layer results.

4.1. Performance of the core-layer

Figure 2 presents the benchmark results for the core-layer functions of ML-KEM-512 and ML-KEM-1024 on the `x86-64-ref` and `aarch64-native` architectures under

[3]Acceleration instruction set are automatically disabled by virtualization in this specific test environment.

varying thread counts. The reported values correspond to the speedup relative to the single-threaded model. For each combination of (i) architecture; (ii) thread count, and (iii) core-layer function, the experiment was repeated 50,000 times after a warm-up phase, and the median execution time was used to compute the speedup as in Equation 1.

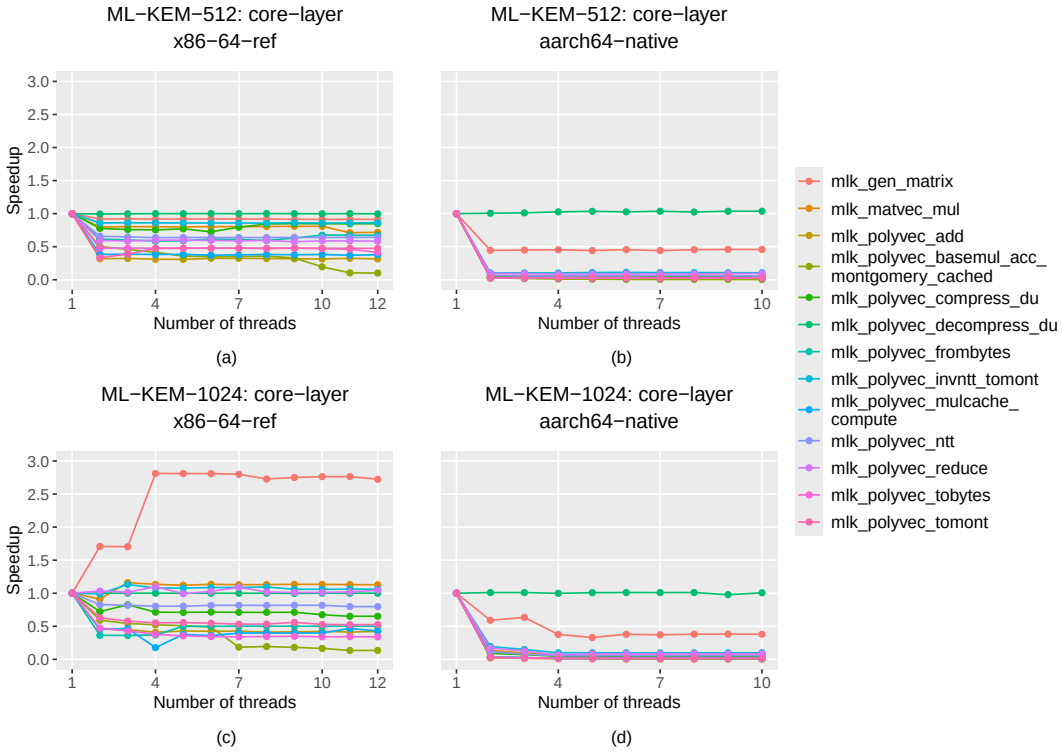


Figure 2. Benchmark results for the core-layer functions of ML-KEM-512 and ML-KEM-1024 on the `x86-64-ref` and `aarch64-native` architectures.

Figure 2 shows similar behavior in plot (a), corresponding to ML-KEM-512 on `x86-64-ref`, and in plots (b) and (d), corresponding to ML-KEM-512 and ML-KEM-1024 on `aarch64-native`, respectively. In all three cases, no function achieved speedup above 1.0, indicating that the overhead introduced by OpenMP was significant relative to the short execution time of the functions. This observation agrees with the literature, which indicates that execution times on the order of tens of microseconds lie near the threshold at which OpenMP can provide meaningful speedup (see [Fredrickson et al. 2003] and Lindberg[4]), and is especially evident for the `aarch64-native` architecture.

In contrast, plot (c) shows that, for ML-KEM-1024 on `x86-64-ref`, the functions `mlk_gen_matrix`, `mlk_matvec_mul`, and `mlk_polyvec_invntt` achieved speedups consistently greater than 1.0. Among them, `mlk_gen_matrix` reached the highest speedup, exceeding 2.5× with four or more threads.

Figure 2 also indicates that the strategy described in Section 3.1 effectively controlled thread-management overhead, since allowing more threads than necessary did not degrade performance.

Since these results were obtained with isolated functions, the speedups may differ

[4]https://software.intel.com/en-us/articles/basic-openmp-threading-overhead

in the complete ML-KEM implementation because of additional parallelization overhead. Therefore, a conservative approach was adopted to avoid degraded performance: in the subsequent analyses, only `mlk_gen_matrix` was parallelized, and only for ML-KEM-1024 on `x86-64-ref`; in all other cases, parallelism remained disabled.

4.2. Performance of the API-layer

Section 4.1 identified the core-layer functions worth parallelizing with OpenMP. As discussed there, only `mlk_gen_matrix` is parallelized, and only for ML-KEM-1024 on `x86-64-ref`, to provide the best performance for `libcesarpq`. In this section, we evaluate the impact of this decision at the API-layer and compare the results with `liboqs`.

Figure 3 presents the API-layer benchmark results for ML-KEM-512 and ML-KEM-1024 on `x86-64-ref` and `aarch64-native` under varying thread counts. The reported values correspond to the speedup relative to the single-threaded model. For each combination of (i) architecture; (ii) thread count; and (iii) API-layer function (i.e., keypair generation, encapsulation, and decapsulation), the experiment was repeated 150,000 times after a warm-up phase, and the median execution time was used to compute the speedup.

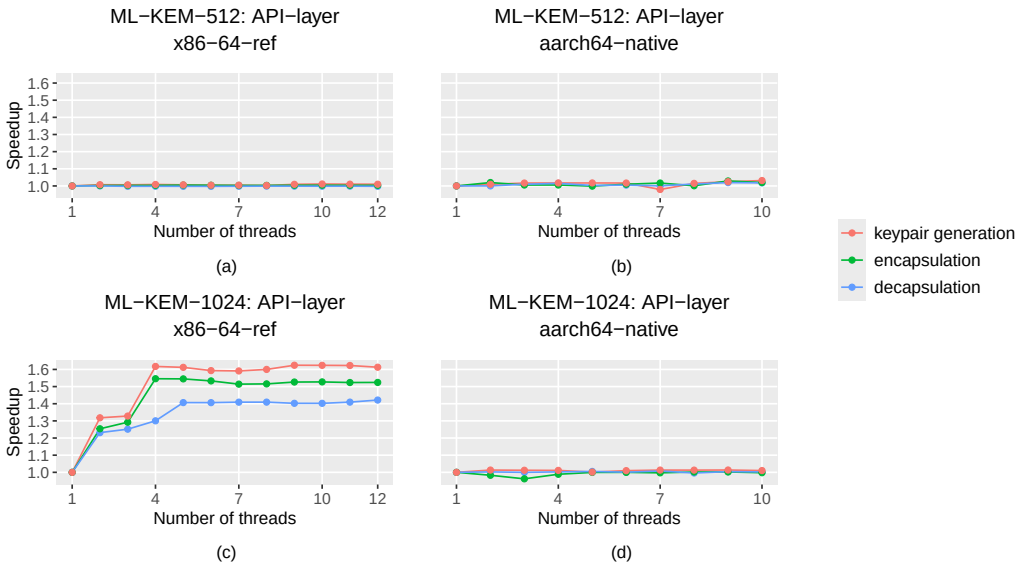


Figure 3. Benchmark results for the API-layer of ML-KEM-512 and ML-KEM-1024 on the `x86-64-ref` and `aarch64-native` architectures.

Consistent with Section 4.1, Figure 3(a), (b), and (d) show no speedup when additional threads are available, since no core-layer function is parallelized for ML-KEM-512 on `x86-64-ref` or for ML-KEM-512 and ML-KEM-1024 on `aarch64-native`. These results indicate that OpenMP introduces no observable overhead when parallelism is disabled.

In contrast, Figure 3(c), corresponding to ML-KEM-1024 on the `x86-64-ref` architecture, shows a speedup ranging from 1.4× to 1.6× for the keypair generation, encapsulation, and decapsulation. These results indicate that the `mlk_gen_matrix` function has a substantial impact on the execution time of the ML-KEM algorithm. When

evaluated in isolation, it achieved a speedup of about 2.5× (see Figure 2(c)); even when combined with the other ML-KEM functions, it still yielded a speedup greater than 1.4× when measured through the API-layer functions.

Table 1 provides another relevant comparison by combining (i) architectures; (ii) single-threaded or multithreaded execution[5]; and (iii) API-layer functions, while also including `liboqs`[6] for comparison. Both libraries were compiled with the most suitable settings for each architecture.

Table 1. Median execution times (μs) and speedup relative to the baseline of each experimental block for API-layer.

Backend	Algorithm	Thread Type	Keypair (μs)	Speedup	Encaps. (μs)	Speedup	Decaps. (μs)	Speedup
Block A: (baseline `libcesarpq-1.5.1-x86-64-ref`, ML-KEM-512, single-thread)								
`libcesarpq-1.5.1-x86-64-ref`	**ML-KEM-512**	**single (baseline)**	**23.213**	**1.000**	**24.966**	**1.000**	**26.997**	**1.000**
`libcesarpq-1.5.1-x86-64-ref`	ML-KEM-512	multi	23.018	1.008	24.928	1.002	27.016	0.999
`liboqs-v0.15.0-x86-64-ref`	ML-KEM-512	native	24.215	0.959	26.631	0.937	26.359	1.024
Block B: (baseline `libcesarpq-1.5.1-aarch64-native`, ML-KEM-512, single-thread)								
`libcesarpq-1.5.1-aarch64-native`	**ML-KEM-512**	**single (baseline)**	**4.792**	**1.000**	**5.625**	**1.000**	**5.875**	**1.000**
`libcesarpq-1.5.1-aarch64-native`	ML-KEM-512	multi	4.750	1.009	5.417	1.038	5.708	1.029
`liboqs-v0.15.0-aarch64-native`	ML-KEM-512	native	7.459	0.642	8.500	0.662	9.875	0.595
Block C: (baseline `libcesarpq-1.5.1-x86-64-ref`, ML-KEM-1024, single-thread)								
`libcesarpq-1.5.1-x86-64-ref`	**ML-KEM-1024**	**single (baseline)**	**59.769**	**1.000**	**63.411**	**1.000**	**64.263**	**1.000**
`libcesarpq-1.5.1-x86-64-ref`	ML-KEM-1024	multi	36.947	1.618	41.459	1.529	45.213	1.421
`liboqs-v0.15.0-x86-64-ref`	ML-KEM-1024	native	59.847	0.999	63.226	1.003	60.978	1.054
Block D: (baseline `libcesarpq-1.5.1-aarch64-native`, ML-KEM-1024, single-thread)								
`libcesarpq-1.5.1-aarch64-native`	**ML-KEM-1024**	**single (baseline)**	**10.333**	**1.000**	**11.417**	**1.000**	**13.208**	**1.000**
`libcesarpq-1.5.1-aarch64-native`	ML-KEM-1024	multi	10.167	1.016	11.416	1.000	13.167	1.003
`liboqs-v0.15.0-aarch64-native`	ML-KEM-1024	native	18.459	0.560	19.750	0.578	23.000	0.574

In Table 1 (Block A), as expected, there is almost no difference between the single-threaded and multithreaded versions of `libcesarpq` for ML-KEM-512 on `x86-64-ref`, consistent with Figure 3(a). Its performance is also very similar to that of `liboqs`, since both use the same backend, namely `mlkem-native`.

For `x86-64-ref` with ML-KEM-1024, shown in Table 1 (Block C), the multithreaded version of `libcesarpq` achieves a speedup of about 1.6× over the single-threaded version, as already discussed in Figure 3(c). This gain is explained by the OpenMP-based parallelization of `mlk_gen_matrix`. As expected, `liboqs` performs similarly to the single-threaded `libcesarpq`, but it is outperformed by its multithreaded version.

For `aarch64-native`, Table 1 (Blocks B and D), the single-threaded and multithreaded versions of `libcesarpq` show similar behavior, in agreement with Figures 3(b) and (d). However, `libcesarpq` is approximately 50% to 80% faster than `liboqs`. This gap is explained by backend specialization and by the build options exposed by each library. In `libcesarpq`, ML-KEM provides a dedicated `aarch64+sha3` path, compiled with `-march=armv8.4-a+sha3` and selected at runtime when SHA3 instruction set architecture (ISA) support is available; in addition, the

[5]In single-thread mode, OpenMP is disabled. In multithreaded mode, `libcesarpq` automatically defines the number of threads to be used.

[6]For `liboqs`, version `0.15.0` was used, as it was the latest stable release at the time this paper was written (https://github.com/open-quantum-safe/liboqs/releases/tag/0.15.0).

release build applies native tuning and interprocedural optimization (IPO)/link-time optimization (LTO). For `liboqs`, we used the best configuration available without source-code changes, enabling its `aarch64-native` ML-KEM path. However, `liboqs` does not currently expose an equivalent `aarch64+sha3` backend selectable only through compilation directives, which explains the observed performance gap.

Finally, it is important to highlight the significant performance improvement observed when architecture-specific optimized implementations are available. In this particular case, the optimized implementation used on `aarch64-native` is 4× to 5× faster than the reference-oriented implementation used on `x86-64-ref`. This result should be interpreted with caution, since it reflects a specific experimental setting and may vary according to factors such as processor architecture, microarchitectural characteristics, compiler optimizations, available instruction-set extensions, and processor model. Even so, it clearly indicates that optimized, architecture-aware implementations can substantially outperform generic reference implementations.

4.3. Performance of the batched API-layer

This section analyzes the batched API-layer. As discussed in Section 3.3, nested parallelism is intentionally disabled, so the batched API-layer invokes the API-layer in single-threaded mode and applies OpenMP only at the upper layer.

Figure 4 presents the batched API-layer benchmark results for ML-KEM-512 and ML-KEM-1024 on `x86-64-ref` and `aarch64-native` under different thread counts. The reported values correspond to the speedup relative to the single-threaded baseline. For each combination of (i) architecture; (ii) thread count; and (iii) batched API-layer function (i.e., batched keypair generation, batched encapsulation, and batched decapsulation), the experiment was repeated 5,000 times after a warm-up phase, and the median execution time was used to compute the speedup. Each benchmark repetition processes a batch of 256 calls to the underlying API-layer; that is, each batch performs 256 executions of keypair generation, encapsulation, or decapsulation, depending on the function being evaluated.

Figure 4 shows similar behavior within the same architecture. In particular, Figure 4(a) and (c), corresponding to ML-KEM-512 and ML-KEM-1024 on `x86-64-ref`, show near-linear speedup, reaching about 12× with 12 threads. According to Equation 2, this indicates that, within the evaluated range, the reduction of the useful-work term, $\frac{T_s}{p}$, remains dominant over the overhead term $T_{oh}(p)$.

In contrast, Figure 4(b) and (d), corresponding to `aarch64-native`, initially show an approximately linear trend, but the speedup soon stabilizes. With 10 threads, the speedup ranges from about 2× to 4× for ML-KEM-512 and from about 4× to 5× for ML-KEM-1024. In terms of Equation 2, this saturation indicates that, as p increases, the reduction in $\frac{T_s}{p}$ is progressively offset by the growth of $T_{oh}(p)$.

This effect is associated with the shorter execution time of the batched operations on `aarch64-native`, which implies a smaller T_s. As a result, the overhead term $T_{oh}(p)$ becomes comparatively more significant as the number of threads p increases, thereby limiting the achievable speedup. This phenomenon is more pronounced for ML-KEM-512 than for ML-KEM-1024 on the same architecture, since the former involves less useful work and is therefore more strongly affected by the relative weight of $T_{oh}(p)$. A

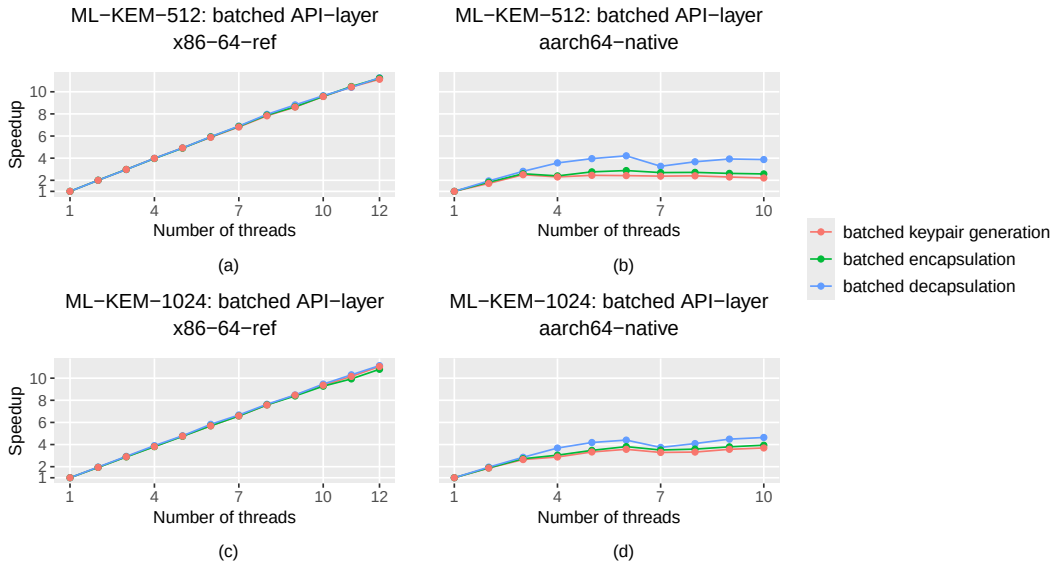


Figure 4. Benchmark results for the batched API-layer of ML-KEM-512 and ML-KEM-1024 on the `x86-64-ref` and `aarch64-native` architectures.

similar effect would also be expected on the `x86-64-ref` architecture; however, within the evaluated range, the larger absolute execution time of the functions implies a larger T_s, so the term $\frac{T_s}{p}$ still dominates over $T_{oh}(p)$, and the saturation effect is not observed.

As discussed in Section 4.2, Table 2 provides another relevant comparison by combining (i) architectures; (ii) single-threaded and multithreaded execution modes; and (iii) batched API-layer functions, while also including `liboqs`. Both libraries were compiled with the most suitable configuration for each target architecture.

To enable a clearer comparison of the batched API-layer results, we adopted as baseline the single-threaded version of `libcesarpq`, with all OpenMP-based parallelization disabled, including at the batched API-layer. We then compared it with three alternatives: (i) a single-threaded version of `libcesarpq` with externally implemented OpenMP-based batching[7]; (ii) a multithreaded version of `libcesarpq` using the internally implemented extended batched API; and (iii) `liboqs` with externally implemented OpenMP-based batching, so as to ensure a fair comparison between the two libraries.

Table 2 can likewise be analyzed by architecture. For `x86-64-ref`, `libcesarpq` and `liboqs` exhibit similar performance, as expected, since both rely on the same `mlkem-native` backend. Moreover, consistently with Figure 4, an approximate 12× speedup is achieved, in line with the near-linear scaling observed up to the 12 available threads.

For `aarch64-native`, the results differ between security levels. For ML-KEM-512, the observed speedup ranges from about 2× to 4×. Although ML-KEM-512 on `aarch64-native` is faster in `libcesarpq` than in `liboqs` in the non-batched baseline comparison (Table 1), the OpenMP overhead in the batched scenario offsets much of this advantage. According to Equation 2, this occurs because the useful-work

[7]In this scenario, we implemented the batching manually in the test code using OpenMP using the single-threaded version of `libcesarpq`.

Table 2. Median execution times per execution (μs) and speedup relative to the baseline of each experimental block for batched API-layer.

Backend	Algorithm	Thread Type	Keypair (μs)	Speedup	Encaps. (μs)	Speedup	Decaps. (μs)	Speedup
Block A: (baseline `libcesarpq-1.5.1-x86-64-ref`, ML-KEM-512, single-thread)								
`libcesarpq-1.5.1-x86-64-ref`	**ML-KEM-512**	**single (baseline)**	**23.349**	**1.000**	**26.023**	**1.000**	**29.382**	**1.000**
`libcesarpq-1.5.1-x86-64-ref`	ML-KEM-512	single (OpenMP external)	2.069	11.284	2.328	11.176	2.591	11.339
`libcesarpq-1.5.1-x86-64-ref`	ML-KEM-512	multi	2.085	11.198	2.290	11.364	2.586	11.360
`liboqs-v0.15.0-x86-64-ref`	ML-KEM-512	native (OpenMP external)	3.316	7.040	3.766	6.909	2.537	11.580
Block B: (baseline `libcesarpq-1.5.1-aarch64-native`, ML-KEM-512, single-thread)								
`libcesarpq-1.5.1-aarch64-native`	**ML-KEM-512**	**single (baseline)**	**4.659**	**1.000**	**5.607**	**1.000**	**5.874**	**1.000**
`libcesarpq-1.5.1-aarch64-native`	ML-KEM-512	single (OpenMP external)	1.992	2.339	2.085	2.689	1.589	3.697
`libcesarpq-1.5.1-aarch64-native`	ML-KEM-512	multi	2.137	2.180	2.182	2.570	1.467	4.003
`liboqs-v0.15.0-aarch64-native`	ML-KEM-512	native (OpenMP external)	1.986	2.346	2.309	2.428	2.531	2.321
Block C: (baseline `libcesarpq-1.5.1-x86-64-ref`, ML-KEM-1024, single-thread)								
`libcesarpq-1.5.1-x86-64-ref`	**ML-KEM-1024**	**single (baseline)**	**60.573**	**1.000**	**65.270**	**1.000**	**71.791**	**1.000**
`libcesarpq-1.5.1-x86-64-ref`	ML-KEM-1024	single (OpenMP external)	5.285	11.462	5.733	11.385	6.279	11.434
`libcesarpq-1.5.1-x86-64-ref`	ML-KEM-1024	multi	5.467	11.079	6.013	10.855	6.414	11.193
`liboqs-v0.15.0-x86-64-ref`	ML-KEM-1024	native (OpenMP external)	5.772	10.493	6.161	10.592	6.011	11.943
Block D: (baseline `libcesarpq-1.5.1-aarch64-native`, ML-KEM-1024, single-thread)								
`libcesarpq-1.5.1-aarch64-native`	**ML-KEM-1024**	**single (baseline)**	**10.169**	**1.000**	**11.462**	**1.000**	**13.273**	**1.000**
`libcesarpq-1.5.1-aarch64-native`	ML-KEM-1024	single (OpenMP external)	2.820	3.607	2.979	3.847	3.178	4.176
`libcesarpq-1.5.1-aarch64-native`	ML-KEM-1024	multi	2.721	3.737	2.916	3.931	2.759	4.811
`liboqs-v0.15.0-aarch64-native`	ML-KEM-1024	native (OpenMP external)	4.189	2.427	4.694	2.442	5.274	2.517

term, $\frac{T_s}{p}$, becomes relatively small, while $T_{oh}(p)$ becomes comparatively more significant. By contrast, for ML-KEM-1024, the larger execution time keeps $\frac{T_s}{p}$ significant relative to $T_{oh}(p)$, so parallel execution still provides a clear benefit. As a result, `libcesarpq` remains slightly faster, achieving speedup from 3.6$\times$ to 4.8$\times$, whereas `liboqs` achieves around 2.5$\times$.

The main takeaway in this scenario concerns product-engineering advantages. With `libcesarpq` in multithreaded mode, the user does not need to implement, maintain, and validate a custom parallelization layer: the library already provides built-in parallelism with a consistent API and well-defined behavior. This reduces development effort, lowers the risk of implementation errors, and decreases maintenance costs while preserving competitive performance.

5. Security considerations

In the context of this work, the use of OpenMP only modifies the execution strategy of the implementation. As such, it does not, by construction, introduce additional side effects beyond those inherent to the original sequential algorithm. The implications of this design choice are discussed in the following.

In the core-layer, parallelization is applied to loops whose iterations operate on disjoint memory regions, preserving polynomial computation independence and avoiding concurrent writes to the same destination. In particular, matrix generation and polynomial/vector NTT-domain operations are index-partitioned so each thread processes independent blocks, while seeds, parameters, and lookup tables are read-only. Therefore, assuming no aliasing among output buffers, parallel execution preserves the same semantics as serial execution.

In the batched API layer, parallelism is applied across full ML-KEM instances (keypair generation, encapsulation, and decapsulation functions) that are independent by construction. Each loop iteration computes its own publick key, private key, ciphertext,

and shared secret offsets and calls the scalar API on non-overlapping slices; thus, no cryptographic state is shared across batch instances. The only shared aggregate in the loop is the return code, handled with OpenMP reduction, which prevents a data race on that variable. In addition, the backend dispatcher is initialized once in a constructor and then only read during parallel calls.

The security justification for OpenMP is therefore conditional on explicit assumptions, consistent with the `libcesarpq` design: i) buffers passed to parallel calls must be disjoint per instance/iteration; ii) shared inputs (e.g., reference public keys or input seeds) must remain read-only during parallel regions; iii) custom extensions (e.g., backend hooks, custom RNG, or external callbacks) must preserve thread safety; and iv) parallel computations at the iteration level operate only on their own inputs and produce outputs without reading or modifying the state of other iterations.

Regarding the risk of running two threads on the same physical core, `libcesarpq` itself enforces thread-placement policy in its thread initializer and, in automatic mode, caps thread count to detected physical cores. Under this operational model, two worker threads are not intended to be scheduled on the same physical core, reducing intra-core contention and constraining side-channel surface related to shared pipeline/L1 resources under Simultaneous Multithreading (SMT). In summary, under the above conditions, OpenMP parallelization is compatible with the implementation's security requirements.

6. Conclusion

In this work, we presented `libcesarpq`, an OpenMP-based cryptographic library for ML-KEM, and evaluated different parallelization strategies across the core-layer, API-layer, and batched API-layer. Our results showed that the impact of parallelism depends strongly on the granularity of the computation: while most internal functions are too short to benefit significantly from fine-grained OpenMP parallelization, the extended batched API-layer enables the library to exploit parallelism much more effectively, yielding the most consistent speedups. When compared with `liboqs`, `libcesarpq` achieved competitive performance overall and outperformed it in some scenarios, especially on optimized platforms. These results indicate that the proposed extended API is particularly advantageous for high-throughput use cases, as it combines usability with efficient batch execution while avoiding nested parallelism overhead. Finally, the paper analyzes the security implications of the proposed OpenMP-based modifications, highlighting that the proposed optimizations preserve the algorithm's role as a practical and secure building block for post-quantum cryptography.

Acknowledgments

This work has been funded by the project PQC Optimization for Critical Infrastructures supported by Centro Integrado de Segurança em Sistemas Avançados (CISSA), with financial resources from the PPI IoT/Manufatura 4.0 of the MCTI grant number 08/2-24, signed with EMBRAPII.

References

[Abbasi et al. 2025] Abbasi, M., Cardoso, F., Váz, P., Silva, J., and Martins, P. (2025). A practical performance benchmark of post-quantum cryptography across heterogeneous computing environments. *Cryptography*, 9(2):32. pages 7

[Avanzi et al. 2019] Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J. M., Schwabe, P., Seiler, G., Stehlé, D., et al. (2019). Crystals-kyber algorithm specifications and supporting documentation. *NIST PQC Round*, 2(4):1–43. pages 2

[Azevedo et al. 2025] Azevedo, B. L., Lagrota, V., and Ribeiro, M. V. (2025). Multicore implementation of ml-kem on embedded devices. In *Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg)*, pages 675–691. SBC. pages 3

[Becker et al. 2022] Becker, H., Hwang, V., Kannwischer, M. J., Yang, B.-Y., and Yang, S.-Y. (2022). Neon ntt: Faster dilithium, kyber, and saber on cortex-a72 and apple m1. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 221–244. pages 2

[Bos et al. 2018] Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J. M., Schwabe, P., Seiler, G., and Stehlé, D. (2018). Crystals-kyber: a cca-secure module-lattice-based kem. In *2018 IEEE European symposium on security and privacy (EuroS&P)*, pages 353–367. IEEE. pages 2

[Chen et al. 2016] Chen, L. et al. (2016). Report on post-quantum cryptography. Technical Report 8105, NIST, Gaithersburg, MD. pages 1

[Fredrickson et al. 2003] Fredrickson, N. R., Afsahi, A., and Qian, Y. (2003). Performance characteristics of openmp constructs, and application benchmarks on a large symmetric multiprocessor. In *Proceedings of the 17th annual international conference on Supercomputing*, pages 140–149. pages 8

[Gewehr et al. 2024] Gewehr, C., Luza, L., and Moraes, F. G. (2024). Hardware acceleration of crystals-kyber in low-complexity embedded systems with risc-v instruction set extensions. *IEEE Access*, 12:94477–94495. pages 2

[Grover 1996] Grover, L. K. (1996). A fast quantum mechanical algorithm for database search. In *Proc. 28th annual ACM symposium on Theory of computing*, pages 212–219. pages 1

[Kim and Seo 2025] Kim, Y. and Seo, S. C. (2025). An optimized instantiation of post-quantum mqtt protocol on 8-bit avr sensor nodes. In *Proceedings of the 20th ACM Asia Conference on Computer and Communications Security*, pages 248–266. pages 2

[Lagrota et al. 2022] Lagrota, V., Camponogara, Â., López, J., and Ribeiro, M. V. (2022). The feasibility of the crystals-kyber scheme for smart metering systems. *IEEE Access*, 10:131303–131317. pages 2

[Lagrota et al. 2025] Lagrota, V., Filomeno, M. d. L., de MBA Dib, L., López, J., and Ribeiro, M. V. (2025). A quantum-resistant advanced metering infrastructure. In *Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg)*, pages 32–48. SBC. pages 2

[Langlois and Stehlé 2015] Langlois, A. and Stehlé, D. (2015). Hardness of the learning with errors problem over rings and modules. In *Advances in Cryptology – EUROCRYPT 2015*, volume 9056 of *Lecture Notes in Computer Science*, pages 464–494. Springer. pages 2

[Nguyen et al. 2025] Nguyen, T.-H., Dang, T.-K., Dam, D.-T., Nguyen, K.-D., Duong, P.-P., Pham, C.-K., and Hoang, T.-T. (2025). An area-time efficient hardware architecture for ml-kem post-quantum cryptography standard. *IEEE Access*. pages 2

[Ni et al. 2025] Ni, Z., Khalid, A., Liu, W., and O'neill, M. (2025). A highly hardware efficient ml-kem accelerator with optimised architectural layers. *ACM Transactions on Embedded Computing Systems*, 24(2):1–24. pages 2

[NIST 2024] NIST (2024). Module-lattice-based key-encapsulation mechanism standard (fips 203). Federal Information Processing Standards Publication FIPS 203, National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA. Published August 13, 2024. pages 2

[Pacheco and Malensek 2021] Pacheco, P. and Malensek, M. (2021). *An introduction to parallel programming*. Morgan Kaufmann. pages 3

[Paiva et al. 2025] Paiva, T. B., Simplicio Jr, M. A., Hafiz, S. M., Yildiz, B., Cominetti, E. L., and Ogawa, H. S. (2025). Tailorable codes for lattice-based kems with applications to compact ml-kem instantiations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(3):139–163. pages 2

[Regev 2005] Regev, O. (2005). On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 84–93. ACM. pages 2

[Shor 1994] Shor, P. W. (1994). Algorithms for quantum computation: Discrete logarithms and factoring. In *Proc. 35th Annu. Symp. Foundations of Computer Science*, pages 124–134. pages 1

[Wei et al. 2026] Wei, H., Li, W., Shen, S., Yang, H., and Zhao, Y. (2026). Optimized implementation of ml-kem on armv9-a with sve2 and sme. *Cryptology ePrint Archive*. pages 2

APKHash: Grafos de Similaridade Estrutural como Evidência de Ruído na Rotulação de Famílias de Malware

Felipe Duarte[1], Eloiza Rossetto[1], João Pincovscy[2,3], Paulo Lisboa de Almeida[1], André Grégio[1]

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)
SecRET (secret.seg.br) – Curitiba – PR – Brasil
{felipeduarte, eloiza.rossetto, paulorla, gregio}@ufpr.br

[2]Centro de Pesquisa e Desenvolvimento para a Segurança das Comunicações
[3]UnDF - Universidade do Distrito Federal
Brasília – DF – Brasil
pincovscy@cepesc.gov.br; joao.pincovscy@undf.edu.br

Abstract. *In this work, we present APKHash, an approach based on opcode n-grams, MinHash/LSH, and similarity graphs for the structural analysis of Android applications. The methodology was evaluated on 12,000 APKs from 24 malware families collected between 2022 and 2024. The results show that heterogeneous components persist even in restrictive settings, indicating shared structural patterns between families or inconsistencies in dataset labeling. Regarding performance, the use of MinHash/LSH as APKHash main approach reduced the search space by approximately 97%. APKHash captures relevant structural relationships and reveals strong intra-family cohesion, but also limitations in separating malware families. The approach proves useful for structural analysis and for diagnosing the quality of large-scale malware datasets labeling.*

Resumo. *Neste trabalho, apresentamos o APKHash, uma abordagem baseada em n-gramas de opcodes, MinHash/LSH e grafos de similaridade para a análise estrutural de aplicações Android. A metodologia foi avaliada em 12.000 APKs de 24 famílias de malware coletadas entre 2022 e 2024. Os resultados mostram que componentes heterogêneos persistem mesmo em configurações restritivas, indicando padrões estruturais compartilhados entre famílias ou inconsistências na rotulagem do conjunto de dados. Em relação ao desempenho, o uso de MinHash/LSH como principal abordagem do APKHash reduziu o espaço de busca em aproximadamente 97%. O APKHash captura relações estruturais relevantes e revela forte coesão intra-família, mas também limitações na separação entre famílias de malware. A abordagem mostrou-se útil tanto para a análise estrutural quanto para o diagnóstico da qualidade da rotulagem de grandes conjuntos de dados de malware.*

1. Introdução

A identificação de famílias de *malware* é uma etapa importante para a área de cibersegurança. Embora o foco geral se volte para a detecção de *malware*, somente a partir do estudo de exemplares é possível compreender os mecanismos explorados por atacantes e criar defesas mais eficazes para sistemas computacionais. Uma família de *malware* pode

ser definida como uma coleção de arquivos maliciosos originários de uma mesma base de código [Joyce et al. 2025]. Determinar a família de um exemplar ou definir novas famílias é um trabalho feito manualmente por analistas de segurança. Após a análise, um "nome"e uma assinatura são criados e posteriormente usados na indústria por antivírus.

Esse processo é oneroso e há conhecimento da área que a rotulação de famílias é muitas vezes inconsistente, como destacado por [Wang et al. 2024]. Não há um consenso de rótulo entre diferentes antivírus e variantes surgem tão rapidamente que analistas não conseguem atualizar assinaturas no mesmo passo. Atualmente, com a popularização e o uso extensivo de algoritmos de aprendizado de máquina, a extração de características para a criação de conjuntos de dados vem se tornando cada vez mais importante. A qualidade dos conjuntos de dados usados para treinamento e validação impacta diretamente a qualidade final dos modelos gerados e, portanto, na capacidade de proteger sistemas computacionais. Nesse cenário, este artigo propõe-se agrupar famílias de *malware* de forma mais representativa quanto a sua similaridade fornecendo insights sobre a qualidade dos rótulos dos exemplares atribuídos a uma dada família.

Para tanto, introduz-se o APKHash, um *pipeline* para representação de APKs baseado em um grafo de similaridade construído a partir de dados dos arquivos `.dex`. Cada APK é representada por padrões locais extraídos com n-gramas refinados e transformados em assinaturas utilizando MinHash. Em seguida, emprega-se *Locality Sensitive Hashing* (LSH) [Paulevé et al. 2010] para agrupar assinaturas semelhantes sem a necessidade de comparar todas as APKs diretamente. O resultado é um grafo de similaridade no qual cada nó representa uma APK e as arestas conectam aplicações cuja similaridade de Jaccard ultrapassa um limiar definido. Como esse grafo é naturalmente desconexo, surgem componentes conectados que podem ser analisados para investigar relações estruturais entre exemplares e validar os rótulos de famílias de *malware*.

O APKHash foi avaliado em um subconjunto do AndroZoo [Allix et al. 2016], utilizando as famílias definidas em [Hurier et al. 2017]. A partir desses dados, foram aplicados filtros para construir um conjunto balanceado contendo apenas famílias com pelo menos 500 amostras, resultando em um *dataset* com 12.000 APKs distribuídos em 24 famílias. Sobre o grafo gerado, conduzimos uma análise dos principais componentes conectados e investigamos relações estruturais entre famílias de *malware* a partir das representações produzidas pelo APKHash.

Os resultados indicam que o método é capaz de gerar representações discriminativas para diferentes famílias de *malware*. A análise dos componentes conectados revelou a presença de famílias com assinaturas mais genéricas, além de possíveis inconsistências nos rótulos atribuídos, sugerindo erros de rotulagem ou sobreposição comportamental entre amostras de *malware*.

Portanto, as principais contribuições trazidas por esse trabalho são: (i) APKHash, um método escalável para a geração de representações estruturais de APKs voltado à análise de similaridade entre aplicações Android; (ii) Análise de componentes gerados pelo método para a validação de rótulos de família de *malwares* Android; (iii) Discussão sobre o comportamento e relação entre famílias de *malware* Android.

2. Trabalhos Relacionados

Na literatura, diferentes trabalhos investigam similaridade entre aplicações Android utilizando informações extraídas do APK, como permissões, chamadas de API, estrutura de classes ou características do bytecode. Entre essas abordagens, os trabalhos [Canfora et al. 2015], [Kang et al. 2016] e [Lee et al. 2019] utilizam n-gramas de *opcodes* extraídos dos arquivos `.dex` para representar aplicações Android. Em [Canfora et al. 2015], os autores analisam a eficácia de sequências de *opcodes* para detecção de *malware*s de múltiplas famílias, mostrando que padrões locais do bytecode possuem capacidade discriminativa. De forma semelhante, [Kang et al. 2016] combina n-gramas de *opcodes* com aprendizado de máquina para classificação de *malware* Android, enquanto o *Dexofuzzy* [Lee et al. 2019] utiliza *fuzzy hashing* sobre sequências de *opcodes* para detectar variantes de *malware*.

Outros trabalhos investigam comparação de aplicações Android completas para detectar clonagem, reutilização de código, variantes ou *repackaging*. O trabalho *AnDarwin* propõe uma abordagem escalável baseada em informação semântica e MinHash/LSH para evitar comparações par a par entre APKs [Crussell et al. 2015]. Já o *RepDetector* [Guan et al. 2016] utiliza relações de entrada e saída entre funções para detectar *repackaging*, buscando robustez frente a transformações que preservam semântica. Trabalhos mais recentes, como [De Ghein et al. 2022] e [Dai et al. 2025], investigam comparação estrutural entre versões de aplicações Android por meio da estrutura de classes e técnicas de *diffing*, permitindo identificar correspondências mesmo na presença de obfuscação. Além disso, técnicas de *similarity hashing* também têm sido utilizadas para detecção de variantes de *malware* Android. O *FLSH* utiliza *fuzzy hashing* sobre características estáticas de APKs para detecção de *malware* e variantes [Hadi et al. 2025].

Há também trabalhos que utilizam grafos e redes de similaridade para modelar relações entre aplicações Android. Em [Frenklach et al. 2021a] e [Frenklach et al. 2021b], grafos de similaridade entre APKs são empregados para apoiar a detecção de *malware*. Já em [Karbab et al. 2020], o *Cypider* constrói uma infraestrutura baseada em redes de similaridade e aplica detecção de comunidades para particionar amostras de *malware* Android. De forma semelhante, [Kim et al. 2019] utiliza redes de similaridade e análise de comunidades para classificar famílias de *malware*. *Frameworks* como o *MATRIX* [Simoni et al. 2025] também utilizam representações baseadas em grafos para apoiar análise de *malware* e investigação de ameaças em larga escala. Trabalhos recentes, como *DeepCatra* [Wu et al. 2023], [Li et al. 2024] e *GHGDroid* [Shen et al. 2024], utilizam grafos heterogêneos, fluxo de execução e características extraídas do bytecode para tarefas de detecção de *malware* Android.

Em geral, a literatura apresenta técnicas para detectar *repackaging*, comparar versões ou aplicações, aplicar *similarity hashing* e utilizar grafos em tarefas relacionadas a *malware* Android. Diferente dessas abordagens, o APKHash combina n-gramas de *opcodes*, MinHash/LSH e grafos de similaridade para analisar como APKs Android se organizam estruturalmente em escala. A representação proposta permite estudar agrupamento, fragmentação e mistura entre famílias de *malware*, possibilitando investigar relações estruturais entre APKs independentemente dos rótulos atribuídos aos conjuntos de dados.

3. Metodologia e Conceitos

Nessa seção, será discutida a metodologia utilizada no APKHash, apresentando-se os conceitos necessários em cada etapa de aplicação da mesma.

3.1. O Conjunto de Dados

Aplicações Android são distribuídas no formato APK, que possui diferentes componentes necessários para a execução da aplicação, como códigos, manifesto, metadados, etc. Dentre esses componentes, os arquivos `.dex` armazenam o *bytecode* Dalvik da aplicação, que servem como entrada para análise do APKHash, pois eles possuem a representação do código executável de apps Android. Essa escolha de representação torna a comparação mais focada no código executável, mas também impõe limites relacionados ao código nativo, recursos, manifesto e comportamento dinâmico não serem integrados na análise. Por outro lado, bibliotecas comuns, empacotadores (ferramentas que comprimem, otimizam e protegem o código), e trechos reutilizados podem estar presentes em diferentes APKs e produzir similaridade entre aplicações que possuem rótulos de famílias diferentes, possibilitando o agrupamento pela estrutura da APK e auxiliando a investigação de rotulação errônea.

APKHash foi avaliado com parte do *dataset* AndroZoo [Allix et al. 2016], um dos maiores conjuntos de dados de *malware* de android disponíveis na literatura. O AndroZoo conta com milhões de APKs benignas e maliciosas. Em [Hurier et al. 2017], os autores selecionam um subconjunto do Androzoo e criam um método (Euphony) para unificar rótulos de famílias de *malware* a partir de detecções de diferentes antivírus. Assim, os autores disponibilizam um *dataset* com mais de 200 mil amostras rotuladas via Euphony.

A partir dessas informações, foi feito o *download* das amostras, as quais foram passadas para uma plataforma de análise de APKs [Júnior et al. 2025]. Essa etapa foi crucial para garantir que as APKs fossem completas e funcionais. Durante o processo de execução dinâmica, muitos exemplares foram descartados por se tratarem apenas de fragmentos de arquivos executáveis. Em diversos casos, não foi possível reconstruir as APKs a partir desses fragmentos. Assim, optou-se pelo descarte dessas amostras, uma vez que impossibilitaria a comparação com APKs completas.

Ao todo, obteve-se 42.230 APKs analisadas dinamicamente, distribuídas em 54 famílias diferentes. Em seguida, aplicamos um novo filtro aos dados, mantendo apenas as famílias com pelo menos 500 exemplares, o que resultou em 24 famílias selecionadas. Além disso, foi estabelecido um limite de 500 APKs por família, com o objetivo de obter uma base de dados balanceada para os experimentos. Dessa forma, a base final é composta por 12.000 amostras pertencentes a 24 famílias, as quais serão listadas após aceite para reprodução dos experimentos feitos neste artigo.

3.2. Extração de n-gramas

A partir dos arquivos `.dex`, são extraídas sequências de *opcodes*, os quais representam a operação executada por uma instrução de *bytecode* como: operações de carga e movimentação, (mover valores literais ou dados entre registradores), operações aritméticas e lógicas, operações de objetos e campos, entre outras. Os *opcodes* são então transformados em 4-gramas, que representam sequências locais compostas por quatro operações consecutivas do *bytecode* da APK. Essa representação mantém a ordem local das instruções e

permite capturar padrões estruturais do código, evitando comparar a sequência completa de *opcodes* que pode ser mais sensível a pequenas modificações no *bytecode*.

Cada n-grama pode ser representado por um identificador numérico, reduzindo o custo de armazenamento e comparação. Dessa forma, cada APK passa a ser tratada como um conjunto de identificadores derivados dos seus n-gramas, o que permite aplicar medidas de similaridade entre conjuntos, como Jaccard, e técnicas de indexação aproximada, como MinHash e LSH.

Por fim, os 4-gramas extraídos são deduplicados para capturar a presença de padrões locais no arquivo, fazendo com que cada APK passe a ser representada por um conjunto quadrigramas únicos. Após a extração, cada conjunto gerado é serializado para reduzir o tamanho de armazenamento utilizando o BLAKE2b [Saarinen and Aumasson 2015]. Com essa etapa cada elemento do conjunto passa a ocupar o tamanho fixo de 8 *bytes*.

3.3. Assinaturas MinHash e Indexação LSH

A comparação exata entre todos os pares de elementos de uma base cresce quadraticamente com o número de amostras, o que pode se tornar inviável para bases grandes. Técnicas como MinHash e *Locality-Sensitive Hashing* (LSH) são utilizadas para reduzir esse custo, recuperando pares candidatos que possuem maior probabilidade de serem similares.

O MinHash é uma técnica utilizada para representar conjuntos por meio de assinaturas compactas. A ideia central é aproximar a similaridade de Jaccard entre conjuntos: quando dois conjuntos compartilham muitos elementos, suas assinaturas MinHash tendem a ser mais parecidas. Dessa forma, a comparação pode ser feita sobre assinaturas menores, em vez de exigir inicialmente a comparação completa entre todos os elementos dos conjuntos originais.

O LSH é utilizado para organizar essas assinaturas de modo que itens semelhantes tenham maior chance de cair nos mesmos grupos de busca, chamados *buckets*. Quando duas amostras aparecem no mesmo *bucket*, elas são tratadas como pares candidatos para uma comparação posterior mais precisa. Assim, o LSH não define sozinho a similaridade final entre duas amostras, mas reduz o espaço de busca para os pares mais promissores.

Portanto, para reduzir o custo das comparações exaustivas na geração de pares de APKs similares, o APKHash utiliza assinaturas MinHash. Cada conjunto de n-gramas extraído das APKs é transformado em uma assinatura de tamanho fixo. Logo, em vez de comparar diretamente os conjuntos completos de n-gramas, o APKHash realiza comparações entre assinaturas compactas, reduzindo o custo computacional. Nos experimentos, foi utilizado com configuração 256 permutações MinHash e semente aleatória 42.

A partir das assinaturas MinHash, aplicou-se a técnica de *Locality-Sensitive Hashing* (LSH) para recuperar pares candidatos de forma eficiente. Para isso, cada assinatura é dividida em b bandas contendo r linhas cada, de modo que $b \cdot r = k$, em que k representa o número total de permutações MinHash. Dois APKs são considerados candidatos quando apresentam coincidência em pelo menos uma das bandas.

Nos experimentos, utilizou-se b=32 e r=8, reduzindo os 71.994.000 pares possíveis a 1.886.845 candidatos únicos, uma redução de 97,38%.

3.4. Construção do Grafo de Similaridade

A partir dos pares candidatos recuperados por LSH, é construído um grafo de similaridade $G = (V, E)$, em que cada nó $v \in V$ representa um APK. Uma aresta $(a, b) \in E$ é adicionada quando o par (a, b) é recuperado pelo LSH e sua similaridade de Jaccard é maior ou igual a um limiar t:

$$(a, b) \in E \iff (a, b) \text{ é candidato LSH} \wedge J(G_a, G_b) \geq t.$$

Foram avaliados os limiares $t \in \{0{,}5, 0{,}6, 0{,}7, 0{,}8, 0{,}9, 1{,}0\}$, abrangendo configurações progressivamente mais restritivas. O valor $t = 0{,}8$ foi selecionado para o estudo de caso por apresentar o maior F1 pareado, enquanto $t = 1{,}0$ foi utilizado como cenário extremo, no qual apenas representações idênticas são conectadas.

O grafo de similaridade gerado pelo APKHash é naturalmente desconexo, isto é, não existe necessariamente um caminho entre todos os pares de vértices do grafo. Dessa forma, parte das análises realizadas considera seus componentes conectados, definidos como subgrafos maximais $C = (V_C, E_C) \subseteq G$ nos quais $\forall u, v \in V_C,\ \exists\, p(u, v)$.

Além disso, os componentes conectados são classificados de acordo com os rótulos de família dos APKs. Seja $f(v)$ a função que retorna a família associada ao vértice v. Um componente é considerado homogêneo quando $|\{f(v) \mid v \in V_C\}| = 1$, enquanto componentes heterogêneos satisfazem $|\{f(v) \mid v \in V_C\}| > 1$.

Essa distinção permite analisar o grau de coesão estrutural entre APKs de uma mesma família, bem como identificar relações estruturais compartilhadas entre famílias distintas.

4. Experimentos e Resultados

Nesta seção são apresentados e discutidos os resultados obtidos com a aplicação do APKHash sobre o conjunto de malware Android analisado. Inicialmente, são avaliados os pares candidatos recuperados pelo MinHash e LSH, incluindo a análise dos *buckets* gerados e da redução do espaço de busca. Em seguida, são investigados os efeitos de diferentes limiares de similaridade de Jaccard na estrutura dos grafos, considerando métricas quantitativas e homogeneidade dos componentes conectados. Finalmente são apresentados estudos de caso sobre componentes mistos gerados em diferentes cenários de similaridade.

4.1. Pares Recuperados pelo LSH

Para avaliar o comportamento da abordagem proposta durante a geração de pares candidatos pelo LSH, o APKHash foi aplicado sobre o conjunto de dados composto por 12.000 APKs distribuídas em 24 famílias de malware (descrito na Seção 3.1). Nesse experimento, foi utilizada uma configuração do MinHash com 256 permutações, organizadas em 32 bandas com 8 linhas por banda. Ao todo, foram gerados 128.467 *buckets*, dos quais 92.479 (72%) continham apenas um item. Os 35.988 *buckets* restantes, contendo mais de um item, apresentaram em média 8 itens agrupados. Entretanto, também foram observados *buckets* anômalos, como o maior deles, que agrupou 640 itens.

A predominância de *buckets* com apenas um item era esperada no uso combinado de MinHash e LSH. Essa baixa taxa de colisão sugere que 72% das assinaturas geradas foram

suficientes para capturar padrões únicos no conjunto de malwares Android analisado. Por outro lado, os *buckets* anômalos com elevado número de colisões podem indicar a presença de trechos comuns entre diferentes APKs, representados pelos 4-gramas extraídos. Esses n-gramas provavelmente correspondem a sequências de opcodes associadas a bibliotecas amplamente utilizadas em aplicações Android. Devido ao funcionamento do MinHash e LSH não é possível reverter as assinaturas dos *buckets* para determinar quais foram os quadrigramas que deram origem a chave.

Em relação às famílias de malware, a grande maioria dos *buckets*, 122.656 (95%), contém amostras de apenas uma única família. Esse resultado era esperado, considerando que 72% dos *buckets* possuíam apenas um item. Mesmo entre os *buckets* com múltiplos itens, observa-se uma elevada taxa de colisões entre amostras da mesma família: 20.177 (56%) dos 35.988 *buckets* com colisões continham exclusivamente APKs de uma única família. Em contrapartida, 15.811 *buckets* apresentaram amostras pertencentes a mais de uma família, representando aproximadamente 12% do total de *buckets* gerados pelo LSH.

Esses *buckets* heterogêneos provavelmente refletem a existência de trechos de código compartilhados entre os malwares Android analisados. Famílias pertencentes a categorias semelhantes, como famílias de trojans, podem reutilizar bibliotecas comuns ou implementar comportamentos similares, resultando em assinaturas parcialmente similares. Ainda assim, embora existam assinaturas do LSH com convergência entre diferentes famílias, elas representam uma porcentagem pequena do total de *buckets* gerados, reforçando a capacidade do método em criar representações de famílias de malware distintas.

Outra forma de avaliar os resultados do MinHash e do LSH é por meio da redução do espaço de busca. O conjunto de dados utilizado possui 71.994.000 pares possíveis. Ao todo, os 35.988 *buckets* com mais de um item produziram 12.024.400 pares. Desse total, existem pares duplicados já que um mesmo par de APKs pode colidir em mais de uma banda. Como cada par deve ser contabilizado uma única vez essas ocorrências foram deduplicadas, resultando em 1.886.845 pares candidatos únicos. Ao considerarmos somente os pares únicos, o APKHash reduziu em aproximadamente 97,38% o número de comparações em relação à busca exaustiva

4.2. Avaliação dos limiares de similaridade

Um dos parâmetros centrais do APKHash é o limiar de similaridade de Jaccard t, utilizado para definir as arestas do grafo. Foram avaliados os valores $t \in \{0{,}5, 0{,}6, 0{,}7, 0{,}8, 0{,}9, 1{,}0\}$, abrangendo configurações progressivamente mais restritivas. Para avaliar os componentes conectados, utilizamos precisão, *recall* e F1 pareados. Um verdadeiro positivo (TP) é um par de APKs da mesma família no mesmo componente; um falso positivo (FP), um par de famílias distintas no mesmo componente; e um falso negativo (FN), um par da mesma família em componentes diferentes. Assim, $P = TP/(TP+FP)$, $R = TP/(TP+FN)$ e $F1 = 2PR/(P+R)$. As métricas pareadas, a pureza e a fração de APKs em componentes mistos consideram as 12.000 amostras, enquanto as métricas de tamanho consideram apenas componentes não unitários. A Tabela 1 resume os resultados.

Na Tabela 1, observa-se que, à medida que o limiar de Jaccard aumenta, o critério para estabelecer conexões entre os nós torna-se mais rigoroso, reduzindo gradualmente o número de nós não isolados, de arestas e o grau médio dos grafos. Em $t = 0{,}5$, o grafo apresenta 11.001 nós, 1.537.433 arestas e grau médio de 279. Já em $t = 0{,}9$, considerando

Tabela 1. Métricas estruturais e de pureza dos grafos para diferentes limiares de Jaccard.

Métrica	**Limiar de Jaccard (`t`)**					
	0.5	**0.6**	**0.7**	**0.8**	**0.9**	**1.0**
Total de Nós	11001	10530	9782	8773	7655	4805
Total de Arestas	1537433	1011265	677026	472525	344373	74204
Grau Médio	279	192	138	107	89	30
Comp. não unitários	222	365	592	837	1071	791
Tamanho Médio Comp.	49.55	28.85	16.52	10.48	7.15	6.07
Maior Comp. (# nós)	7356	6234	4451	622	464	168
Maior Comp. (%)	66%	59%	45%	7%	6%	3%
Componentes Mistos	66	93	135	158	178	126
Precisão Pareada	6.37%	7.22%	9.73%	50.79%	81.32%	84.24%
Recall Pareado	58.98%	48.37%	34.35%	15.97%	10.84%	2.09%
F1 Pareado	11.50%	12.57%	15.16%	24.30%	19.13%	4.07%
Pureza Majoritária	32.77%	43.13%	56.78%	81.83%	92.25%	95.59%
APKs em Comp. Mistos	85.43%	77.36%	66.52%	49.95%	29.07%	14.41%

a faixa não extrema dos limiares avaliados, o grafo possui 7.655 nós e 344.373 arestas. Isso corresponde a reduções de 30,4% no número de nós não isolados e de 77,6% no total de arestas. O grau médio também diminui 68,1%, passando de 279 para 89 conexões por nó. Esses resultados indicam que limiares mais elevados produzem grafos mais esparsos e seletivos, preservando relações de similaridade estrutural mais fortes entre as APKs.

À medida que o limiar de similaridade aumenta o número de nós, arestas e o grau médio dos grafos seguem uma tendência de redução, enquanto os componentes conectados tornam-se menores e mais numerosos. Em `t=0.5`, o grafo possui 222 componentes conectados, enquanto em `t=0.9` esse número aumenta para 1071 componentes. Apesar do crescimento na quantidade total de componentes, cada componente passa a agrupar menos nós, dado que o tamanho médio dos componentes reduz de 49.55 para 7.15 nós. De forma semelhante, o maior componente conectado também diminui significativamente conforme o limiar se torna mais restritivo, indicando uma fragmentação progressiva da estrutura do grafo. Além disso, embora a quantidade absoluta de componentes mistos aumente com limiares mais elevados, sua proporção em relação ao total de componentes apresenta uma redução contínua ao longo dos experimentos. Esse comportamento sugere que componentes contendo APKs de famílias distintas tornam-se relativamente menos frequentes em cenários mais restritivos. Dessa forma, o aumento do limiar de Jaccard resulta em grafos compostos por componentes menores e mais homogêneos, indicando um aumento na precisão do agrupamento de exemplares pertencentes à mesma família.

Para analisar o cenário mais restritivo, avaliamos $t = 1{,}0$, no qual uma aresta é criada apenas entre APKs com conjuntos idênticos de 4-gramas. O grafo resultante contém 4.805 nós e 74.204 arestas, e seu maior componente reúne 168 APKs. Ainda assim, 126 dos 791 componentes são heterogêneos, indicando que representações idênticas também ocorrem entre rótulos distintos. Esse resultado pode decorrer de reutilização de código,

bibliotecas ou empacotadores compartilhados, granularidade dos rótulos ou limitações da representação adotada, não constituindo, isoladamente, evidência de erro de rotulação.

Além das métricas estruturais, a Tabela 1 apresenta métricas pareadas dos componentes conectados. Com o aumento do limiar, a precisão pareada cresce, enquanto o *recall* diminui devido à maior fragmentação das famílias. O maior F1 foi obtido em $t = 0{,}8$, com 24,30%, indicando o melhor equilíbrio observado entre precisão e *recall*. À medida que o limiar de similaridade aumenta, os agrupamentos tornam-se mais homogêneos: a precisão por família cresce de 38,03% para 84,29%, a pureza majoritária aumenta de 32,77% para 92,25% e o percentual de APKs em componentes mistos cai de 85,43% para 29,07%. Em contrapartida, o recall se deteriora gradativamente, atingindo 2,09% em `t=1.0`, comportamento também refletido na redução do F1-Score. Em conjunto com o aumento no número de componentes conectados e a diminuição de seus tamanhos, esses resultados indicam maior fragmentação entre as famílias. Assim, as APKs passam a ocupar componentes menores e mais isolados porém com mais itens pertencentes a mesma família.

Os resultados observados nos experimentos relacionados ao limiar de similaridade de Jaccard mostram que a presença de componentes mistos persiste mesmo em limiares mais restritivos. Esse comportamento reforça a hipótese de que diferentes famílias de malware Android compartilham características estruturais semelhantes, seja pelo reaproveitamento de bibliotecas e trechos de código, seja pela implementação de comportamentos parecidos. Outra possibilidade é a existência de inconsistências nas rotulações fornecidas pelos mecanismos antivírus. Como consequência, APKs pertencentes a famílias distintas podem apresentar assinaturas estruturalmente similares quando representadas pelo APKHash.

5. Estudos de Caso: Componentes Conectados

Para avaliar o APKHash e investigar possíveis ruídos de rotulação, analisamos os componentes conectados nos limiares $t = 0{,}8$ e $t = 1{,}0$. O valor $t = 0{,}8$ apresentou o maior F1 pareado, representando o melhor equilíbrio observado entre precisão e *recall*, além de preservar conexões suficientes para uma análise estrutural mais próxima de um cenário prático. Já $t = 1{,}0$ representa o cenário mais restritivo, no qual apenas APKs com conjuntos idênticos de 4-gramas são conectadas. A permanência de famílias distintas nesses componentes permite investigar indícios de reutilização estrutural, compartilhamento de código ou inconsistências nos rótulos.

5.1. Componentes em t=0.8

Para analisar o cenário `t=0.80`, calculamos métricas considerando os componentes homogêneos e heterogêneos gerados pelo APKHash. A Tabela 2 sumariza essas métricas. O APKHash produziu 837 componentes conectados que agrupam 8.773 APKs, correspondendo a aproximadamente 73% de todo o conjunto de dados avaliado. As APKs restantes não formaram grupos com outras amostras e, portanto, ficaram fora dessa representação, já que o grafo exige ao menos uma aresta considerando o limiar de similaridade de Jaccard maior ou igual a 0.80. Dos componentes conectados gerados, 679 (81%) são homogêneos e os 158 restantes são heterogêneos. Embora a maioria dos componentes seja composta por APKs de uma única família, apenas 31% das APKs do conjunto estão presentes nesses componentes.

Os componentes homogêneos tendem a ser significativamente menores que os heterogêneos. Enquanto os componentes homogêneos apresentam tamanho médio de aproximadamente quatro nós, os componentes mistos possuem média de cerca de 38 nós. Além disso, os componentes heterogêneos apresentam grau médio maior, com aproximadamente 21 conexões por nó, enquanto os homogêneos possuem média de apenas três conexões. Esse comportamento sugere que os componentes mistos concentram APKs altamente conectadas, formando regiões densas do grafo nas quais múltiplas famílias compartilham características estruturais ou comportamentais semelhantes.

Apesar da presença de diferentes famílias nos componentes heterogêneos, a maior parte das conexões ainda ocorre entre APKs da mesma família. Das 445 mil arestas presentes nesses componentes, aproximadamente 343 mil (77%) correspondem a conexões intra-família, enquanto apenas 23% representam conexões entre famílias distintas. Esse resultado indica que, mesmo quando o APKHash produz componentes mistos, ainda há uma forte preservação de similaridade semântica entre as amostras agrupadas. As conexões inter-família podem refletir diferentes cenários, como reutilização de código, compartilhamento de bibliotecas e empacotadores, similaridade comportamental entre famílias distintas ou até inconsistências nas *labels* do conjunto de dados. Além disso, a presença de componentes heterogêneos grandes e densamente conectados sugere que o limiar de Jaccard igual a 0.80 ainda permite a formação de agrupamentos amplos, conectando famílias que apresentam padrões estruturais ou comportamentais semelhantes.

Tabela 2. Métricas dos componentes homogêneos e heterogêneos para o grafo com limiar de Jaccard 0.8

Métrica	Homogêneos	Heterogêneos
Número de componentes	679	158
Total de APKs	2.779	5.994
Percentual de APKs	31.68%	68.32%
Média de tamanho dos componentes	4.09 ± 8.42	37.94 ± 100.79
Grau médio	2.94 ± 8.06	21.29 ± 62.67
Número médio de famílias	1.00	2.89
Total de arestas	26.780	445.745
Arestas intra-família	26.780 (100%)	342.646 (76.87%)
Arestas inter-família	0 (0%)	103.099 (23.13%)

5.1.1. Análise (maiores componentes conectados)

Para complementar a análise da seção anterior, considerando o limiar `t= 0.8`, foram selecionados os cinco maiores componentes conectados homogêneos e heterogêneos para compreender de forma mais profunda como o APKHash está agrupando as famílias de *malware* e de observar mais de perto a estrutura formadas. A partir dos subconjuntos selecionados, gerou-se as visualizações dos grafos, que são apresentadas nas Figuras 1 e 2. Além disso, as métricas quantitativas relacionando esses componentes as famílias foram calculadas (Tabela 3).

Ao observar a Figura 1, é possível identificar grafos densos e altamente interconectados. Todos os grafos gerados, com exceção da família *mailbomb*, seguem esse

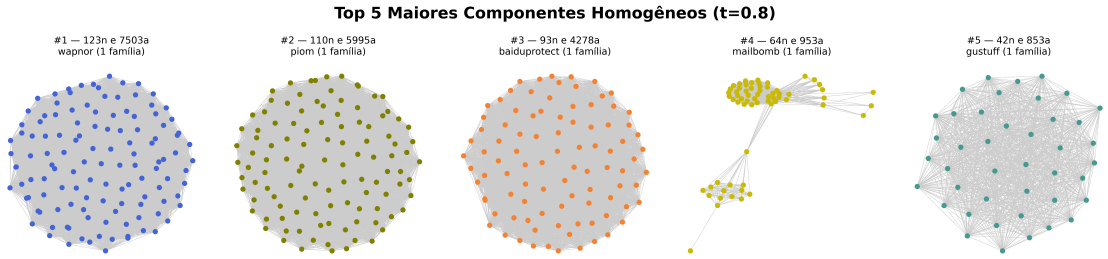


Figura 1. Maiores componentes conectados homogêneos coloridos por família utilizando t=0.8 para similaridade de Jaccard

Tabela 3. Estatísticas dos maiores componentes homogêneos e heterogêneos dos grafos de famílias.

Componentes Homogêneos (top 5 famílias)						
Família	**Nós**	**Arestas/Nó**	**Grau Médio**	**Densidade**		
wapnor	123	61	122	1.000		
piom	110	54	109	1.000		
baiduprotect	93	46	92	1.000		
mailbomb	64	14	29	0.473		
gustuff	42	20	40	0.991		
Componentes Heterogêneos (top 5 famílias)						
Família	**Nós**	**Arestas/Nó**	**Grau Médio**	**Densidade**	**% Família Maj.**	**Nº Famílias**
fakeapp	622	11	22	0.035	18.6	16
packaging*	592	196	393	0.666	63.7	7
cellspy	499	215	431	0.867	96.4	6
scamapp	438	197	394	0.903	90.9	5
gustuff	400	19	39	0.099	45.0	6

padrão de forte conectividade, apresentando em média 39 conexões por APK. Essa elevada conectividade também é refletida na métrica de densidade, na qual todas as famílias, exceto *mailbomb*, apresentam valor igual a 1.0. Esses resultados indicam que, para essas famílias, há uma assinatura bem definida e comportamentos estruturais consistentes.

Para a família *mailbomb*, na Figura 1, gráfico #4, observa-se uma densidade menor em comparação às demais famílias homogêneas. Uma análise do grafo revela a presença de um nó mais distante do *cluster* principal, que se conecta a outras APKs de forma mais dispersa. Esse nó atua como um ponto de ponte entre subestruturas, sugerindo a existência de uma possível variante do *malware*. A partir dessa variante inicial, novos nós passam a se conectar de forma progressivamente mais distribuída, afastando-se do núcleo mais denso do grafo. Nesse contexto, a representação fornecida pelo APKHash permite não apenas identificar famílias bem definidas, mas também evidenciar possíveis variantes dentro de uma mesma família de *malware*.

A Figura 2 apresenta a visualização dos maiores componentes heterogêneos gerados

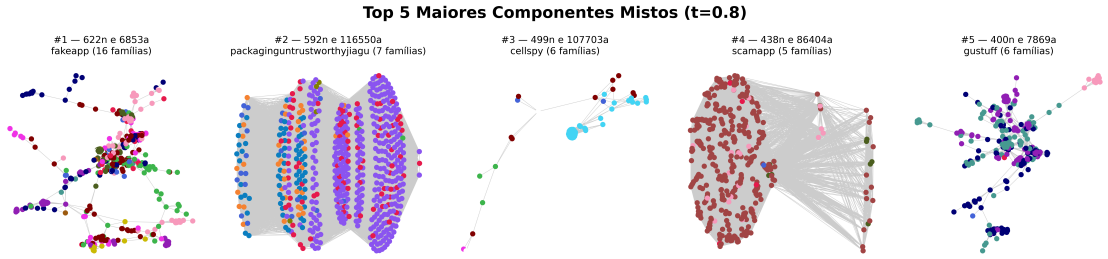


Figura 2. Maiores componentes conectados heterogêneos coloridos por família utilizando t=0.8 para similaridade de Jaccard

com limiar `t=0.8`. Observa-se uma variação significativa tanto na estrutura quanto na densidade dos grafos. Em particular, os componentes associados às famílias *fakeapp* e *gustuff* apresentam estruturas mais esparsas, com menor número de conexões entre as APKs agrupadas. O componente associado à família *fakeapp* se destaca por agrupar mais de 17 famílias distintas, sendo o mais diverso entre os componentes heterogêneos. Essa elevada diversidade se reflete diretamente em sua baixa densidade, que é a menor entre todos os componentes analisados.

Embora o componente de *gustuff* apresente um número menor de famílias em comparação ao de *fakeapp*, sua densidade permanece igualmente baixa. Esse comportamento está relacionado à baixa predominância da família majoritária dentro do componente. Para *fakeapp*, a família dominante representa apenas 18.6% das amostras, enquanto em *gustuff* essa proporção é de 45%, ainda assim insuficiente para formar uma estrutura densamente conectada, uma vez que menos da metade das APKs pertence à mesma família.

Dessa forma, as assinaturas extraídas para as APKs rotuladas como *fakeapp* e *gustuff* apresentam um comportamento mais genérico, indicando sobreposição de características com múltiplas famílias de *malware*. Esse resultado sugere que tais rótulos podem representar agrupamentos amplos de comportamentos maliciosos, em vez de famílias estritamente bem definidas. Nesse contexto, é importante destacar que sistemas de detecção baseados em assinaturas podem, em casos de baixa confiança ou alta ambiguidade, atribuir rótulos genéricos que englobam diferentes padrões de comportamento malicioso. Assim, a análise dos componentes heterogêneos com baixa conectividade entre as amostras permite identificar casos de generalização de assinatura.

Os grafos gerados para as famílias *packaginguntrustworthyjiagu*, *cellspy* e *scamapp* apresentam regiões altamente densas, com forte conectividade entre as APKs. Esses componentes construídos com o APKHash exibem os maiores percentuais de dominância da família majoritária. Em particular, o componente associado à família *cellspy* abrange seis famílias distintas, porém 96.4% das APKs pertencem ao rótulo *cellspy*. Em termos absolutos, das 499 APKs agrupadas, 479 são rotuladas como *cellspy*, enquanto apenas 20 pertencem às demais cinco famílias presentes no componente.

Devido a essa forte predominância, é possível levantar hipóteses acerca da confiabilidade dos rótulos atribuídos às amostras minoritárias. Essas APKs apresentam alta similaridade com a família *cellspy*, o que pode indicar que sejam variantes do mesmo *malware*, possivelmente rotuladas incorretamente ou classificadas sob famílias com assinaturas menos bem definidas.

Outro fator que pode explicar esse comportamento é a presença de nós intermediários (ou "nós ponte") no grafo de *cellspy*. Observa-se uma região altamente conectada e densa, acompanhada de algumas conexões mais periféricas que ligam subestruturas distintas. Esses nós de transição podem representar variantes do *malware* original, especialmente considerando que outras APKs rotuladas como *cellspy* estão localizadas em sua vizinhança no grafo, reforçando a hipótese de continuidade comportamental entre essas amostras.

5.2. Relacionamentos entre famílias

Para compreender melhor o comportamento dos rótulos de famílias de *malware*, realizamos um experimento com limiar de Jaccard igual a 1.0, o valor mais restritivo, no qual apenas APKs com assinatura idêntica pelo APKHash são conectados. Esperava-se que esse cenário eliminasse componentes heterogêneos, porém, como discutido na Seção 4.2, eles ainda foram observados. A partir desses resultados, geramos uma visualização das relações entre famílias, apresentada na Figura 3, na qual a espessura das arestas indica a intensidade das conexões entre elas, permitindo identificar possíveis inconsistências na rotulagem.

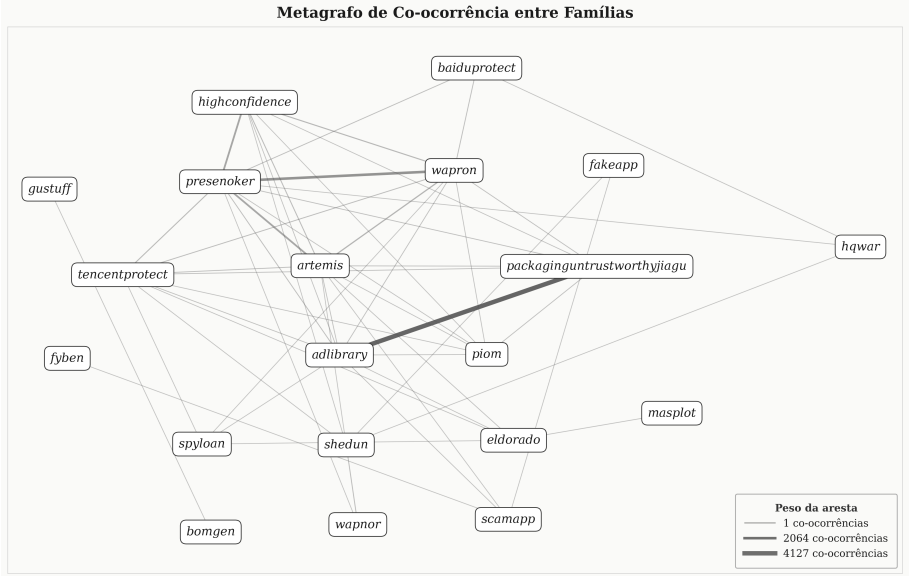


Figura 3. Co-ocorrência entre famílias. Quanto mais espessa as arestas maior é a quantidade de arestas que as famílias conectadas compartilham

Ao analisar o metagrafo apresentado na Figura 3, é possível identificar dois nós com alta centralidade, *adlibrary* e *presenoker*. Ambos apresentam forte conectividade com a maioria das demais famílias, exibindo arestas com pesos elevados, chegando a 4.127 co-ocorrências no par de maior co-ocorrência. Esse comportamento indica que essas famílias podem atuar como categorias mais genéricas no processo de rotulação de *malwares*, agregando APKs cujo padrão comportamental não é suficientemente específico para ser associado a uma família mais especializada.

Em contraste, famílias como *gustuff*, *masplot*, *spyloan* e *fyben* apresentam graus mais baixos e conexões de menor peso, o que sugere comportamentos mais bem definidos e menor sobreposição com outras classes do conjunto. Além disso, a elevada densidade de conexões envolvendo *adlibrary*, *presenoker*, *wapron* e *packaginguntrustworthyjiagu* indica a existência de uma região do espaço de comportamentos em que as fronteiras

entre famílias são pouco nítidas. Esse cenário pode influenciar diretamente o processo de rotulação dessas famílias, pois os comportamentos capturados pela assinatura são comuns a *malwares* e portanto dificultam a caracterização de comportamentos que definem essas famílias.

6. Considerações Finais e Limitações

A ferramenta APKHash para gerar grafos de similaridade e componentes conectados proposta no presente trabalho se mostrou capaz de produzir grafos com forte coesão intra-família, especialmente em limiares mais altos de similaridade, onde a maior parte das conexões ocorre entre APKs com o mesmo rótulo. Ainda assim, mesmo em configurações mais restritivas, incluindo o cenário extremo de `t=1.0`, observam-se componentes conectados mistos, o que sugere a presença de padrões estruturais compartilhados entre famílias distintas ou possíveis inconsistências nos rótulos utilizados nas bases de dados.

Apesar desses achados, a abordagem apresenta algumas limitações, tais como perda de informação estrutural no processo de extração de n-gramas e a geração de assinaturas pode introduzir perdas de informação estrutural, enquanto que a etapa de indexação via LSH pode gerar pares candidatos incorretos, afetando a construção final dos grafos. Além disso, o uso de MinHash/LSH impede a rastreabilidade direta aos n-gramas originais que originaram as similaridades, reduzindo a interpretabilidade fina dos resultados. Ainda assim, essas limitações não invalidam a aplicação do APKHash como uma ferramenta de análise estrutural em larga escala, complementar a abordagens de classificação e rotulação de *malware*, dado que mesmo em cenários restritivos, pode-se observar famílias de *malware* com exemplares muito similares em famílias distintas.

7. Conclusão

Este trabalho apresentou o APKHash, uma abordagem baseada em n-gramas de *opcodes*, MinHash/LSH e grafos de similaridade para analisar relações estruturais entre aplicações Android. A proposta foi motivada pela necessidade de investigar a consistência dos rótulos de famílias de *malware*, frequentemente definidos de forma manual e sujeitos a inconsistências entre diferentes fontes. Ao modelar APKs como grafos de similaridade, foi possível estudar tanto o agrupamento intra-família quanto a presença de conexões entre famílias distintas em larga escala. Os resultados experimentais indicam que o APKHash é capaz de capturar padrões estruturais relevantes de arquivos `.dex`, produzir grafos de similaridade que correlacionam comportamentos intra-família e inter-família. Mesmo em cenários com mais componentes conectados heterogêneos, o APKHash ainda possui a maioria das arestas conectando APKs de mesmo rótulo. Entretanto, durante os experimentos, a persistência de componentes mistos, mesmo em limiares elevados `(t=1.0)`, evidencia limitações importantes, relacionadas tanto ao compartilhamento de código entre diferentes famílias quanto a possíveis inconsistências nos rótulos fornecidos por ferramentas de detecção. Por fim, o estudo mostra que o aumento do limiar de similaridade torna os grafos mais esparsos e homogêneos, mas não elimina completamente a sobreposição entre famílias. Isso reforça a hipótese de que parte das relações observadas não se deve apenas à similaridade de código, mas também à natureza dos próprios rótulos de *malware*. Assim, o APKHash se mostra útil como uma ferramenta de análise estrutural e de diagnóstico da qualidade de *datasets* de *malware*, contribuindo para estudos futuros em curadoria de dados e construção de modelos mais robustos.

AGRADECIMENTOS

Este trabalho foi apoiado pelo Centro de Pesquisa e Desenvolvimento para a Segurança das Comunicações, bem como pelo CNPq por meio de bolsa de produtividade em pesquisa.

Referências

Allix, K., Bissyandé, T. F., Klein, J., and Traon, Y. L. (2016). AndroZoo: Collecting millions of Android apps for the research community. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 468–471. ACM.

Canfora, G., De Lorenzo, A., Medvet, E., Mercaldo, F., and Visaggio, C. A. (2015). Effectiveness of opcode ngrams for the detection of multi family android malware.

Crussell, J., Gibler, C., and Chen, H. (2015). Andarwin: Scalable detection of android application clones based on semantics. *IEEE Transactions on Mobile Computing*, 14(10):2007–2019.

Dai, J., Luo, M., Zhang, Y., Yang, M., and Yang, M. (2025). Apkdiffer: Accurate and scalable cross-version diffing analysis for android applications. *Proc. ACM Program. Lang.*, 9(OOPSLA2).

De Ghein, R., Abrath, B., De Sutter, B., and Coppens, B. (2022). Apkdiff: Matching android app versions based on class structure. In *Proceedings of the 2022 ACM Workshop on Research on Offensive and Defensive Techniques in the Context of Man At The End (MATE) Attacks*, Checkmate '22, page 1–12, New York, NY, USA. Association for Computing Machinery.

Frenklach, T., Cohen, D., Shabtai, A., and Puzis, R. (2021a). Android malware detection via an app similarity graph. *Computers & Security*, 109:102386.

Frenklach, T., Cohen, D., Shabtai, A., and Puzis, R. (2021b). Android malware detection via an app similarity graph. *Computers Security*, 109:102386.

Guan, Q., Huang, H., Luo, W., and Zhu, S. (2016). Semantics-based repackaging detection for mobile apps. In *Engineering Secure Software and Systems*, volume 9639 of *Lecture Notes in Computer Science*, pages 89–105. Springer.

Hadi, H. J., Khalid, A., Hussain, F. B., Ahmad, N., and Alshara, M. A. (2025). FLSH: A framework leveraging similarity hashing for android malware and variant detection. *IEEE Access*.

Hurier, M., Suarez-Tangil, G., Dash, S. K., Bissyandé, T. F., Traon, Y. L., Klein, J., and Cavallaro, L. (2017). Euphony: Harmonious unification of cacophonous anti-virus vendor labels for android malware. In *Proceedings of the 14th International Conference on Mining Software Repositories*, pages 425–435. IEEE Press.

Joyce, R. J., Everett, D., Fuchs, M., Raff, E., and Holt, J. (2025). Claravy: A tool for scalable and accurate malware family labeling. In *Companion Proceedings of the ACM on Web Conference 2025*, pages 277–286.

Júnior, C. T., Filho, D. F., Pincovscy, J., and Grégio, A. (2025). Artemis: Uma plataforma modular para execução, monitoração e investigação de aplicativos android suspeitos. In

Anais do XXV Simpósio Brasileiro de Cibersegurança, pages 147–162, Porto Alegre, RS, Brasil. SBC.

Kang, B., Yerima, S. Y., Sezer, S., and McLaughlin, K. (2016). N-gram opcode analysis for android malware detection. *International Journal on Cyber Situational Awareness*, 1(1):231–254.

Karbab, E. B., Debbabi, M., Derhab, A., and Mouheb, D. (2020). Scalable and robust unsupervised android malware fingerprinting using community-based network partitioning. *Computers & Security*, 96:101932.

Kim, H. M., Song, H. M., Seo, J. W., and Kim, H. K. (2019). ANDRO-SIMNET: Android malware family classification using social network analysis.

Lee, S., Jung, W., Kim, S., Lee, J., and Kim, J.-S. (2019). Dexofuzzy: Android malware similarity clustering method using opcode sequence. Virus Bulletin.

Li, T., Shou, P., Wan, X., Li, Q., Wang, R., Jia, C., and Xiao, Y. (2024). A fast malware detection model based on heterogeneous graph similarity search. *Computer Networks*, 254:110799.

Paulevé, L., Jégou, H., and Amsaleg, L. (2010). Locality sensitive hashing: A comparison of hash function types and querying mechanisms. *Pattern Recognition Letters*, 31(11):1348–1358.

Saarinen, M. J. and Aumasson, J. P. (2015). The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC). RFC 7693.

Shen, L., Fang, M., and Xu, J. (2024). Ghgdroid: Global heterogeneous graph-based android malware detection. *Computers Security*, 141:103846.

Simoni, M., Saracino, A., et al. (2025). Matrix: A comprehensive graph-based framework for malware analysis and threat research. In *Proceedings of the 22nd International Conference on Security and Cryptography, SECRYPT*, pages 11–13.

Wang, L., Wang, H., Zhang, T., Xu, H., Meng, G., Gao, P., Wei, C., and Wang, Y. (2024). Android malware family labeling: Perspectives from the industry. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ASE '24, page 2176–2186, New York, NY, USA. Association for Computing Machinery.

Wu, Y., Shi, J., Wang, P., Zeng, D., and Sun, C. (2023). Deepcatra: Learning flow- and graph-based behaviours for android malware detection. *IET Information Security*, 17(1):118–130.

msgX: Rumo a Mensagens Pós-Quânticas Soberanas para Implantações Governamentais do Protocolo Matrix

Vinícius Lagrota[1], Henrique Viana[2], Kauã Maia[2], Marcos Ortiz[2], Windson Viana[2], Gilvan Maia[2], Paulo Rego[2], Rodrigo Pacheco[1], Rossana Andrade[2], José Antônio Macêdo[2]

[1] Centro de Pesquisa e Desenvolvimento para a Segurança das Comunicações (CEPESC)
Brasília, DF – Brasil.

[2]Universidade Federal do Ceará (UFC) – Fortaleza, CE – Brazil

Abstract. *This paper presents msgX, a secure messaging protocol for replacing Olm-based session distribution in governmental Matrix rooms, combining standardized and government-developed cryptography to support interoperability and digital sovereignty. To mitigate quantum threats, msgX integrates Post-Quantum Extended Diffie–Hellman (PQXDH), a modified Double Ratchet, and msgGX, a Megolm adaptation for scalable group encryption. In 14,040 Android runs across two devices, 30 each, and 234 parametrized configurations, a branch-level cost decomposition against Olm was performed, isolating the msgX-*Γ *and msgX-*Σ *profiles. The results show that each profile keeps new-member key agreement at* $O(1)$*, preserves stable per-message latency, and incurs a one-time session-establishment overhead of approximately* $+38\%$ *(msgX-*Γ*) and* $+53\%$ *(msgX-*Σ*) relative to Olm. The protocol enables gradual migration from Olm without breaking Matrix federation.*

Resumo. *Este artigo apresenta o msgX, um protocolo de mensagens seguras para substituir a distribuição de sessões baseada no Olm em salas Matrix governamentais, combinando criptografia padronizada e desenvolvida pelo governo para apoiar interoperabilidade e soberania digital. Para mitigar ameaças quânticas, o msgX integra PQXDH, um Double Ratchet modificado e o msgGX, uma adaptação do Megolm para criptografia escalável em grupos. Em 14.040 execuções Android, distribuídas em dois dispositivos, 30 repetições cada e 234 configurações parametrizadas, foi realizada uma decomposição de custo por ramo em relação ao Olm, isolando os perfis msgX-*Γ *e msgX-*Σ*. Os resultados mostram que cada perfil mantém o acordo de chaves para novos membros em* $O(1)$*, preserva latência por mensagem estável e apresenta sobrecarga única de estabelecimento de sessão de aproximadamente* $+38\%$ *(msgX-*Γ*) e* $+53\%$ *(msgX-*Σ*) em relação ao Olm. O protocolo permite migração gradual a partir do Olm sem romper a federação Matrix.*

1. Introdução

Mensagens seguras evoluíram de sistemas de bate-papo na Internet em texto claro para plataformas de criptografia de ponta a ponta (E2EE) que protegem as comunicações mesmo em ambientes de rede hostis (e.g., Signal). Contudo, modelos modernos de segurança enfrentam a perspectiva de computadores quânticos. Mecanismos de chave pública, como Rivest-Shamir-Adleman (RSA) e criptografia de curvas elípticas (ECC), baseiam-se

em hipóteses de dificuldade vulneráveis ao algoritmo de Shor [Shor 1994]. Em resposta, o processo de criptografia pós-quântica (PQ) do Instituto Nacional de Padrões e Tecnologia (NIST) levou à padronização de novos algoritmos, incluindo o CRYSTALS-Kyber, padronizado como ML-KEM na FIPS-203 [Nist 2024]. Embora o computador quântico criptanaliticamente relevante (CRQC) ainda não esteja disponível, a ameaça de coletar agora, descriptografar depois (HNDL) torna urgente a migração, pois o tráfego criptografado capturado hoje pode ser descriptografado no futuro. Assim, a proteção PQ é essencial para o estabelecimento de chaves e para a confidencialidade das mensagens.

Protocolos modernos de mensagens seguras já iniciaram essa transição. Por exemplo, o Signal introduziu o Extended Diffie–Hellman Pós-Quântico (PQXDH) [Kret and Schmidt 2023], que amplia o Diffie–Hellman de curvas elípticas (ECDH) clássico com um mecanismo de encapsulamento de chaves (KEM) PQ, sugerindo que construções híbridas podem melhorar a resiliência durante o período de transição, preservando a segurança ainda que a componente clássica ou PQ seja posteriormente enfraquecida. Contudo, criptografia isoladamente é insuficiente nas comunicações governamentais. Soberania [1], controle da infraestrutura e independência em relação a provedores estrangeiros de tecnologia também são requisitos centrais. Esse contexto levou ao desenvolvimento de iniciativas soberanas de mensageria segura baseadas no Matrix [Matrix.org 2024], um protocolo aberto e descentralizado de comunicação, como o Tchap [Française 2019].

Este artigo apresenta o msgX, um protocolo criptográfico para a distribuição de sessões em salas Matrix para mensagens governamentais de alta garantia, adotando um projeto híbrido e de dois ramos. O ramo padrão utiliza algoritmos amplamente implantados, incluindo mecanismos PQ alinhados aos esforços atuais de padronização, para preservar a interoperabilidade com clientes e federações Matrix. Seu ramo governamental incorpora primitivas criptográficas soberanas, implementadas por meio da *libharpia* [Pacheco et al. 2022], para fortalecer a resiliência e a soberania digital.

As contribuições deste artigo são três: (i) o projeto do msgX, um protocolo criptográfico de dois ramos que substitui o Olm em implantações governamentais do Matrix, preservando a compatibilidade retroativa com clientes existentes; (ii) a especificação do msgX-PQXDH e do msgX-ratchet, bem como sua integração com o protocolo de mensagens em grupo msgGX; e (iii) uma avaliação experimental por decomposição de custo por ramo em hardware Android de uso corrente, isolando os perfis msgX-Γ e msgX-Σ em relação ao Olm e mostrando que os resultados são consistentes com a preservação dos custos $O(N)$ para distribuição de chaves de grupo e $O(1)$ para entrada de novos membros do Megolm, ao mesmo tempo que cada perfil adiciona um fator pós-quântico constante.

2. Fundamentação

A arquitetura dos protocolos modernos de mensagens seguras foi fortemente influenciada pelo Signal e posteriormente incorporada a sistemas como o Matrix. O Extended Triple Diffie-Hellman (X3DH) [Marlinspike and Perrin 2016] permite o estabelecimento assíncrono de sessões a partir do pacote de pré-chaves do destinatário, mesmo quando ele está *offline*. Em seguida, o Double Ratchet (DR) [Perrin and Marlinspike 2016] evolui a

[1]Usamos soberania digital como a capacidade de uma implantação nacional controlar, auditar e substituir as primitivas criptográficas usadas para comunicação, evitando dependência de um único fornecedor, implantação ou autoridade.

chave raiz, as chaves de cadeia e as chaves de mensagem. Esse processo contribui para o sigilo direto (FS) ao limitar a exposição de mensagens anteriores e para a segurança pós-comprometimento (PCS) ao permitir a recuperação da segurança após a incorporação de novo material secreto. Para enfrentar ameaças quânticas, o PQXDH [Kret and Schmidt 2023] combina ECDH clássico com um KEM pós-quântico e deriva o segredo da sessão a partir das duas contribuições, reduzindo a exposição ao risco de HNDL.

Esses mecanismos também fundamentam a E2EE do Matrix, protocolo descentralizado e federado no qual usuários de diferentes servidores podem se comunicar sem uma autoridade central [Matrix.org 2024]. O Megolm protege o plano de dados de todas as salas criptografadas, inclusive chats entre dois usuários, por meio de uma sessão de remetente por dispositivo. O Olm, baseado no DR do Signal, atua no plano de controle ao estabelecer canais entre dispositivos para distribuir o material das sessões Megolm. Após essa distribuição, cada mensagem pode ser cifrada uma única vez e publicada para todos os participantes [Matrix.org 2019, Matrix.org 2022]. A renovação periódica dessas sessões limita a exposição futura após comprometimentos. A Vodozemac implementa em Rust os protocolos Olm e Megolm usados no ecossistema Matrix [Matrix.org Foundation]. Embora modernize a base criptográfica dos clientes e preserve a retrocompatibilidade das sessões, a biblioteca não possui mecanismos PQ para acordo ou renovação de chaves.

Em um contexto mais amplo de mensageria segura, diferentes propostas têm investigado mecanismos de renovação PQ. O Triple Ratchet [Dodis et al. 2025], por exemplo, distribui material criptográfico PQ ao longo de várias mensagens com o uso de códigos de apagamento. Essa abordagem reduz a sobrecarga de banda e aumenta a robustez em fluxos de comunicação desbalanceados, embora introduza maior complexidade de sincronização e gerenciamento de estado.

Outros trabalhos concentram-se na avaliação e otimização dessa transição. Auerbach et al. [Auerbach et al. 2025] propõem uma metodologia baseada em Sparse Continuous Key Agreement (SCKA) e na métrica *Vulnerable Message Set* (VMS), que quantifica as mensagens expostas após o comprometimento de um estado criptográfico, fundamentandoo *ML-KEM Braid* do Signal [Schmidt and Connell 2025], que fragmenta e transmite progressivamente o material PQ em ambientes com restrição de banda. Esses trabalhos evidenciam o compromisso entre frequência de renovação, volume de dados e recuperação após comprometimentos.

No contexto de grupos, o Messaging Layer Security (MLS), padronizado pela IETF no RFC 9420 [Barnes et al. 2023], organiza os participantes em uma árvore criptográfica e faz o estado do grupo avançar por épocas. Inclusões, remoções e atualizações são consolidadas por mensagens de compromisso, permitindo FS e PCS com custos de atualização logarítmicos. A comunidade Matrix investiga o MLS como uma possível evolução de sua criptografia de grupos [Hodgson and Chathi 2023]. Essa integração enfrenta um desafio de conciliar a sequência linear de épocas do MLS com o complexo ambiente federado do Matrix. Em paralelo, a IETF desenvolve suítes PQ e híbridas para MLS baseadas em ML-KEM [Mahy and Barnes 2026].

Essas iniciativas representam trajetórias complementares para a evolução das mensagens seguras. O msgX atua na distribuição e renovação de sessões de grupo em implantações governamentais federadas, com o objetivo de preservar a interoperabilidade com

o Matrix, incorporar mecanismos criptográficos soberanos e reduzir a exposição ao risco de HNDL no ecossistema. Para isso, substitui os canais Olm usados no transporte do material de sessão por mecanismos híbridos PQ, mas mantém o modelo operacional do Megolm. Assim, preserva a distribuição de chaves de grupo em $O(N)$ e a entrada de novos membros em $O(1)$ sob a camada PQ adicionada, oferecendo uma trajetória incremental de migração com sobrecarga constante.

3. O Protocolo msgX

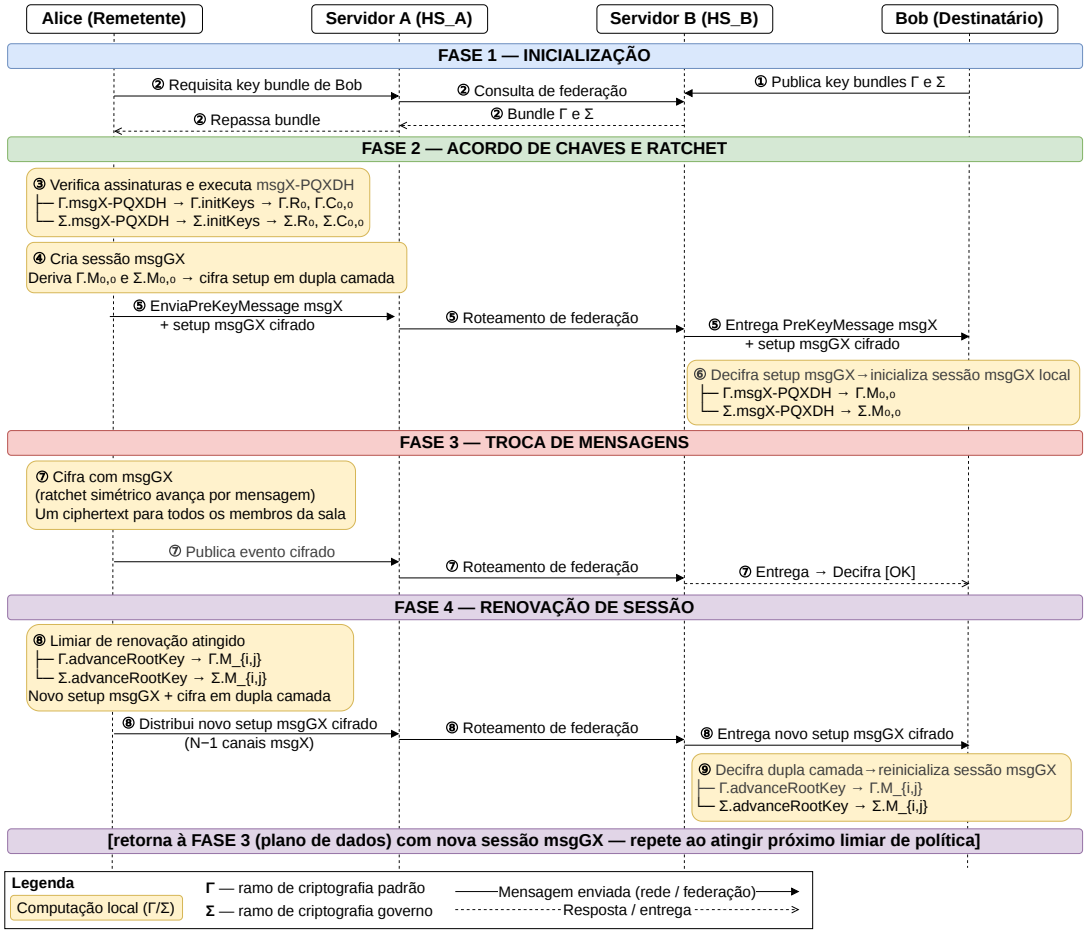


Figura 1. Fluxo fim a fim dos protocolos msgX e msgGX.

Embora o msgX introduza uma arquitetura distinta, seu desenvolvimento é orientado por ideias centrais do PQXDH e do Olm. Ao longo desta seção, Γ denota o ramo de criptografia padrão e Σ o ramo desenvolvido pelo governo. Todas as chaves, algoritmos e procedimentos recebem prefixos correspondentes. Antes de detalhar a estrutura interna do msgX, é importante explicar como ele se integra às salas Matrix e ao seu protocolo complementar de mensagens em grupo, o msgGX.

No Matrix, conversas criptografadas são salas, incluindo chats 1:1; portanto, o msgX mira o cenário de grupo ao proteger a distribuição e a renovação do material de sessão de grupo. O tráfego de grupo de alto volume é criptografado pelo msgGX, uma

adaptação do protocolo Megolm com quatro modificações principais: (i) dois *ratchets* simétricos paralelos, um por ramo, em vez de um único ratchet; (ii) criptografia autenticada com dados associados (AEAD) em dupla camada usando *message keys* específicas de cada ramo; (iii) constantes Hash-based Message Authentication Code (HMAC) separadas por domínio para garantir que as duas cadeias evoluam de forma independente; e (iv) distribuição de sessão por canais msgX em vez de Olm, possibilitando o estabelecimento de sessão PQ descrito na Seção 3.1. Isso preserva a comunicação eficiente em grupo, mas mantém o compromisso do Megolm: o comprometimento de um estado simétrico de grupo atual pode afetar mensagens futuras até a próxima renovação de sessão.

Para mitigar essa limitação, msgX e msgGX são usados em conjunto. O msgGX renova periodicamente sua sessão ao gerar um *setup* inicial novo, que deve ser entregue com segurança a cada participante da sala. O msgX fornece esse canal de entrega: ele transmite o *setup* da sessão msgGX individualmente a cada membro da sala. Nesse projeto, o msgX protege a distribuição da sessão de grupo, enquanto o msgGX criptografa as mensagens de grupo efetivamente trocadas na sala. Uma visão geral da interação entre um remetente e um membro da sala (i.e., Alice e Bob) é apresentada na Figura 1; a mesma etapa de distribuição é repetida para cada membro da sala. Como ilustrado, o msgX é estruturado em quatro fases principais: inicialização; acordo de chaves e ratchet; troca de mensagens; e renovação de sessão.

3.1. Inicialização

Esta fase envolve a geração e publicação antecipada das chaves públicas de Bob, que gera vários pares de chaves e envia a parte pública de cada um ao seu homeserver(vide etapa de setup ❶ na Figura 1). Para suportar tanto Γ quanto Σ, Bob gera dois pares de identity keys: $\Gamma.\mathsf{IK}_B$ e $\Sigma.\mathsf{IK}_B$. Ele também gera um conjunto de pares de ephemeral keys: $\Gamma.\mathsf{EK}_B$, $\Sigma.\mathsf{EK}_B$, e ephemeral keys pós-quânticas $\Gamma.\mathsf{PQE}_B$ e $\Sigma.\mathsf{PQE}_B$. Todas as chaves associadas à criptografia padrão são assinadas usando a identity key privada de Bob $\Gamma.\mathsf{IK}_B^{priv}$, enquanto as chaves governamentais são assinadas com $\Sigma.\mathsf{IK}_B^{priv}$. Note-se que identity keys fornecem uma identidade criptográfica de longo prazo, permitindo autenticação e estabelecimento de confiança. Ephemeral keys, por sua vez, são usadas para FS, garantindo que, mesmo se chaves de longo prazo forem comprometidas posteriormente, chaves de sessões passadas permaneçam seguras.

Bob então cria um key bundle composto pelas partes públicas de todas as chaves geradas, seus identificadores e assinaturas correspondentes. Esse key bundle é enviado ao homeserver, onde fica disponível para recuperação por outros usuários ❷. Isso permite que Alice inicie um acordo de chaves com Bob mesmo quando ele está offline, viabilizando o estabelecimento assíncrono e seguro de sessão.

3.2. Acordo de Chaves e Ratchet

Esta fase começa quando Alice recupera o key bundle de Bob a partir do homeserver. Ela verifica cada assinatura digital incluída no bundle usando $\Gamma.\mathsf{IK}_B^{pub}$ para a criptografia padrão e $\Sigma.\mathsf{IK}_B^{pub}$ para a criptografia governamental. Se qualquer verificação falhar, o protocolo é imediatamente abortado para evitar uma potencial violação de segurança. Uma vez que todas as assinaturas sejam verificadas com sucesso, Alice executa o msgX-PQXDH, como indicado em ❸. O procedimento completo é descrito na Figura 2.

Cada ramo do msgX-PQXDH, isto é, o ramo padrão (Γ) e o ramo governamental (Σ), realiza um encapsulamento KEM, denotado apenas por Enc na Figura 2, e três operações ECDH. O encapsulamento KEM produz um ciphertext e um shared secret para cada ramo, denotados por $\Gamma.\mathsf{ct}, \Gamma.\mathsf{ss}$ e $\Sigma.\mathsf{ct}, \Sigma.\mathsf{ss}$, respectivamente. As operações ECDH envolvem combinar as chaves privadas de Alice com as chaves públicas de Bob, usando identity keys e ephemeral keys. Como resultado, cada ECDH produz três shared secrets adicionais por ramo: $\Gamma.\mathsf{DH}_1, \Gamma.\mathsf{DH}_2, \Gamma.\mathsf{DH}_3$ e $\Sigma.\mathsf{DH}_1, \Sigma.\mathsf{DH}_2, \Sigma.\mathsf{DH}_3$. Para cada ramo, o shared secret do encapsulamento e os três segredos derivados por ECDH são concatenados para formar os segredos combinados $\Gamma.\mathsf{S}$ e $\Sigma.\mathsf{S}$. Esses valores são então usados como entradas para a HKDF dentro do procedimento msgX-ratchet.initKeys($\cdot$), responsável por derivar as root keys e chain keys iniciais do msgX-ratchet: $\Gamma.\mathsf{R}_0, \Gamma.\mathsf{C}_{0,0}$ e $\Sigma.\mathsf{R}_0, \Sigma.\mathsf{C}_{0,0}$.

Figura 2. Protocolo msgX.

Γ.msgX-PQXDH($\Gamma.\mathsf{IK}_A^{priv}, \Gamma.\mathsf{EK}_A^{priv}, \Gamma.\mathsf{EK}_B^{pub}, \Gamma.\mathsf{PQE}_B^{pub}$)

1 : $\Gamma.\mathsf{ss}, \Gamma.\mathsf{ct} \leftarrow \Gamma.\mathsf{Enc}(\Gamma.\mathsf{PQE}_B^{pub})$
2 : $\Gamma.\mathsf{DH}_1 \leftarrow \Gamma.\mathsf{ECDH}(\Gamma.\mathsf{IK}_A^{priv}, \Gamma.\mathsf{EK}_B^{pub})$
3 : $\Gamma.\mathsf{DH}_2 \leftarrow \Gamma.\mathsf{ECDH}(\Gamma.\mathsf{EK}_A^{priv}, \Gamma.\mathsf{IK}_B^{pub})$
4 : $\Gamma.\mathsf{DH}_3 \leftarrow \Gamma.\mathsf{ECDH}(\Gamma.\mathsf{EK}_A^{priv}, \Gamma.\mathsf{EK}_B^{pub})$
5 : $\Gamma.\mathsf{S} \leftarrow \Gamma.\mathsf{DH}_1 || \Gamma.\mathsf{DH}_2 || \Gamma.\mathsf{DH}_3 || \Gamma.\mathsf{ss}$

Σ.msgX-PQXDH($\Sigma.\mathsf{IK}_A^{priv}, \Sigma.\mathsf{EK}_A^{priv}, \Sigma.\mathsf{EK}_B^{pub}, \Sigma.\mathsf{PQE}_B^{pub}$)

1 : $\Sigma.\mathsf{ss}, \Sigma.\mathsf{ct} \leftarrow \Sigma.\mathsf{Enc}(\Sigma.\mathsf{PQE}_B^{pub})$
2 : $\Sigma.\mathsf{DH}_1 \leftarrow \Sigma.\mathsf{ECDH}(\Sigma.\mathsf{IK}_A^{priv}, \Sigma.\mathsf{EK}_B^{pub})$
3 : $\Sigma.\mathsf{DH}_2 \leftarrow \Sigma.\mathsf{ECDH}(\Sigma.\mathsf{EK}_A^{priv}, \Sigma.\mathsf{IK}_B^{pub})$
4 : $\Sigma.\mathsf{DH}_3 \leftarrow \Sigma.\mathsf{ECDH}(\Sigma.\mathsf{EK}_A^{priv}, \Sigma.\mathsf{EK}_B^{pub})$
5 : $\Sigma.\mathsf{S} \leftarrow \Sigma.\mathsf{DH}_1 || \Sigma.\mathsf{DH}_2 || \Sigma.\mathsf{DH}_3 || \Sigma.\mathsf{ss}$

Γ.msgX-ratchet.initKeys($\Gamma.\mathsf{S}$)

1 : $\Gamma.\mathsf{R}_0 || \Gamma.\mathsf{C}_{0,0} \leftarrow \mathsf{HKDF}(0, \Gamma.\mathsf{S}, \text{“OLM\_ROOT”}, 64)$

Σ.msgX-ratchet.initKeys($\Sigma.\mathsf{S}$)

1 : $\Sigma.\mathsf{R}_0 || \Sigma.\mathsf{C}_{0,0} \leftarrow \mathsf{HKDF}(0, \Sigma.\mathsf{S}, \text{“GC\_ROOT”}, 64)$

A sessão msgGX é então criada ❹. Seu setup inicial é criptografado sob a sessão msgX estabelecida, permitindo que Bob reconstrua o mesmo estado msgGX. As root keys e chain keys previamente derivadas produzem duas chaves de criptografia de mensagens, $\Gamma.\mathsf{M}_{0,0}$ e $\Sigma.\mathsf{M}_{0,0}$, usadas com AEAD nos ramos PQ. O setup inicial criptografado da sessão msgGX, juntamente com as chaves públicas usadas por Alice no protocolo msg-PQXDH e os ciphertexts $\Gamma.\mathsf{ct}$ e $\Sigma.\mathsf{ct}$, é enviado a Bob ❺. Do seu lado, Bob recebe e processa a mensagem ❻. Usando os dados recebidos, Bob executa o protocolo msgX-PQXDH e deriva os mesmos shared secrets combinados que Alice. A partir deles, ele obtém as mesmas root keys e chain keys do ratchet para a sessão msgX padrão e governamental. Com isso, Bob consegue derivar as chaves de criptografia de mensagens $\Gamma.\mathsf{M}_{0,0}$ e $\Sigma.\mathsf{M}_{0,0}$ usadas por Alice para criptografar o setup inicial da sessão msgGX. Em seguida, Bob descriptografa o setup inicial e inicializa sua própria sessão msgGX.

3.3. Troca de Mensagens

Os membros da sala trocam mensagens via msgGX, que avança seu próprio ratchet a cada nova mensagem ❼. A sessão msgGX deve ser renovada após um número predeterminado de mensagens trocadas ou um intervalo de tempo especificado, i.e., um novo setup inicial para a sessão msgGX deve ser gerado e transmitido com segurança aos membros da sala.

3.4. Renovação de Sessão

A sessão msgX já estabelecida é reutilizada para renovar a sessão msgGX.No entanto, antes de criptografar o novo setup inicial, o msgX-ratchet deve ser avançado ❽. Isso envolve atualizar as root keys e chain keys para derivar uma nova chave de criptografia de mensagens, que será usada para transmitir com segurança o novo setup inicial da sessão msgGX. A Figura 3 apresenta o msgX-ratchet.advanceRootKey(·), um procedimento para avançar a root key e derivar uma nova chain key. A abordagem proposta se baseia no algoritmo DR [Perrin and Marlinspike 2016], com alguns aprimoramentos introduzidos para atender aos requisitos de segurança e projeto do msgX.

Figura 3. Avançando as root keys no msgX-ratchet.

Γ.msgX-ratchet.advanceRootKey($\Gamma.\mathsf{R}_{i-1}, \Gamma.\mathsf{T}_{i-1}^{priv}, \Gamma.\mathsf{T}_{i}^{pub}, \Gamma.\mathsf{ct}_i$)

1 : $\Gamma.\mathsf{ss}_i \leftarrow \mathbf{null}$
2 : **if** $\exists n \mid i = 2^N n - 1$ **then**
3 : $\quad(\Gamma.\mathsf{PQT}_i^{pub}, \Gamma.\mathsf{PQT}_i^{priv}) \leftarrow \Gamma.\mathsf{KGen}()$
4 : **else if** $\exists n \mid i = 2^N n$ **then**
5 : $\quad\Gamma.\mathsf{ss}_i \leftarrow \Gamma.\mathsf{Dec}(\Gamma.\mathsf{ct}_i, \Gamma.\mathsf{PQT}_i^{priv})$
6 : **endif**
7 : $\Gamma.\mathsf{R}_i || \Gamma.\mathsf{C}_{i,0} \leftarrow \mathsf{HKDF}(\Gamma.\mathsf{R}_{i-1}, \Gamma.\mathsf{ECDH}(\Gamma.\mathsf{T}_{i-1}^{priv}, \Gamma.\mathsf{T}_{i}^{pub}) || \Gamma.\mathsf{ss}_i, \text{“OLM\_RATCHET”}, 64)$

Σ.msgX-ratchet.advanceRootKey($\Sigma.\mathsf{R}_{i-1}, \Sigma.\mathsf{T}_{i-1}^{priv}, \Sigma.\mathsf{T}_{i}^{pub}, \Sigma.\mathsf{ct}_i$)

1 : $\Sigma.\mathsf{ss}_i \leftarrow \mathbf{null}$
2 : **if** $\exists n \mid i = 2^N n - 1$ **then**
3 : $\quad(\Sigma.\mathsf{PQT}_i^{pub}, \Sigma.\mathsf{PQT}_i^{priv}) \leftarrow \Sigma.\mathsf{KGen}()$
4 : **else if** $\exists n \mid i = 2^N n$ **then**
5 : $\quad\Sigma.\mathsf{ss}_i \leftarrow \Sigma.\mathsf{Dec}(\Sigma.\mathsf{ct}_i, \Sigma.\mathsf{PQT}_i^{priv})$
6 : **endif**
7 : $\Sigma.\mathsf{R}_i || \Sigma.\mathsf{C}_{i,0} \leftarrow \mathsf{HKDF}(\Sigma.\mathsf{R}_{i-1}, \Sigma.\mathsf{ECDH}(\Sigma.\mathsf{T}_{i-1}^{priv}, \Sigma.\mathsf{T}_{i}^{pub}) || \Sigma.\mathsf{ss}_i, \text{“GC\_RATCHET”}, 64)$

O algoritmo da Figura 3 define o procedimento para atualizar as root keys e derivar chain keys em ambos os ramos criptográficos, padrão e governamental, sendo invocado sempre que uma renovação da sessão msgGX é necessária. Recebe-se como entrada as root keys anteriores $\Gamma.\mathsf{R}_{i-1}$ e $\Sigma.\mathsf{R}_{i-1}$, as ephemeral keys anteriores e atuais $\Gamma.\mathsf{T}_{i-1}, \Gamma.\mathsf{T}_i$ e $\Sigma.\mathsf{T}_{i-1}, \Sigma.\mathsf{T}_i$, juntamente com os ciphertexts $c\Gamma.\mathsf{T}_i$ e $c\Sigma.\mathsf{T}_i$, que podem conter segredos encapsulados PQ.

Dois tipos de operação são realizados dependendo do índice de etapa atual i. Se i corresponde ao padrão $2^N n - 1$, em que $N \in \mathbb{Z}$ é um valor predeterminado, novos pares de chaves pós-quânticas ($\Gamma.\mathsf{PQT}_i^{pub}, \Gamma.\mathsf{PQT}_i^{priv}$ e $\Sigma.\mathsf{PQT}_i^{pub}, \Sigma.\mathsf{PQT}_i^{priv}$) são gerados para cada ramo, e seus componentes públicos são enviados ao par anexados à mensagem msgX atual. Se $i = 2^N n$, os ciphertexts $c\Gamma.\mathsf{T}_i$ e $c\Sigma.\mathsf{T}_i$ são recebidos anexados à mensagem msgX e decapsulados usando as chaves privadas correspondentes para recuperar os shared secrets PQ $\Gamma.\mathsf{ss}_i$ e $\Sigma.\mathsf{ss}_i$. Esses segredos são usados apenas em intervalos específicos para injetar periodicamente entropia PQ no processo de evolução das chaves.

Por fim, as novas root keys e chain keys $\Gamma.\mathsf{R}_i, \Gamma.\mathsf{C}_{i,0}, \Sigma.\mathsf{R}_i, \Sigma.\mathsf{C}_{i,0}$ são derivadas usando uma HKDF, que recebe como entrada as root keys anteriores $\Gamma.\mathsf{R}_{i-1}$ e $\Sigma.\mathsf{R}_{i-1}$, o resultado de duas operações ECDH separadas, o shared secret PQ opcional $\Gamma.\mathsf{ss}_i$ e $\Sigma.\mathsf{ss}_i$, e um rótulo específico de domínio. Uma operação usa criptografia padrão e a outra usa criptografia desenvolvida pelo governo, entre as ephemeral keys relevantes. Esse mecanismo mantém FS e habilita PCS contra adversários clássicos, ao mesmo tempo que fornece PCS parcial na presença de atacantes com capacidade quântica. Quanto menor o valor de N, mais rápido é o efeito de “healing” em caso de ataque quântico.

Observe que o algoritmo executado pelo destinatário difere ligeiramente. No índice $i = 2^N n$, em vez de gerar um par de chaves ou realizar decapsulamento, o destinatário deve realizar um encapsulamento KEM para derivar os shared secrets $\Gamma.\mathsf{ss}_i$ e $\Sigma.\mathsf{ss}_i$, usando as chaves públicas recebidas $\Gamma.\mathsf{PQT}_i^{pub}$ e $\Sigma.\mathsf{PQT}_i^{pub}$, respectivamente. O restante do algoritmo permanece inalterado. Se a mesma parte que renovou anteriormente a sessão msgGX atual tentar renová-la novamente, ela não terá recebido uma nova ephemeral key do outro par (isto é, $\Gamma.\mathsf{T}_i^{pub}$ e $\Sigma.\mathsf{T}_i^{pub}$) e, portanto, não poderá avançar as root keys. Nesse caso, apenas as chain keys são avançadas usando o procedimento msgX-ratchet.advanceChainKey(·), conforme descrito na Figura 4.

Para avançar apenas as chain keys $\Gamma.\mathsf{C}_{i,j}$ e $\Sigma.\mathsf{C}_{i,j}$, as chain keys anteriores $\Gamma.\mathsf{C}_{i,j-1}$ e $\Sigma.\mathsf{C}_{i,j-1}$ são usadas como chaves em HMACs separados, com um valor fixo usado como mensagem de entrada. Uma vez atualizadas as chain keys, as message keys correspondentes também podem ser derivadas seguindo um procedimento semelhante, o msgX-ratchet.deriveMessageKey(·). As novas message keys $\Gamma.\mathsf{M}_{i,j}$ e $\Sigma.\mathsf{M}_{i,j}$ são geradas usando HMACs (Figura 5), em que as chain keys atualizadas $\Gamma.\mathsf{C}_{i,j}$ e $\Sigma.\mathsf{C}_{i,j}$ servem como chaves dos HMACs e um valor fixo é usado como mensagem de entrada.

Figura 4. Evolução das chain keys no msgX-ratchet.

Γ.msgX-ratchet.advanceChainKey($\Gamma.\mathsf{C}_{i,j-1}$)	Σ.msgX-ratchet.advanceChainKey($\Sigma.\mathsf{C}_{i,j-1}$)
1 : $\Gamma.\mathsf{C}_{i,j} \leftarrow \mathsf{HMAC}(\Gamma.\mathsf{C}_{i,j-1}, 0\text{x}2)$	1 : $\Sigma.\mathsf{C}_{i,j} \leftarrow \mathsf{HMAC}(\Sigma.\mathsf{C}_{i,j-1}, 0\text{x}4)$

Figura 5. Derivação das message keys no msgX-ratchet.

Γ.msgX-ratchet.deriveMessageKey($\Gamma.\mathsf{C}_{i,j}$)	Σ.msgX-ratchet.deriveMessageKey($\Sigma.\mathsf{C}_{i,j}$)
1 : $\Gamma.\mathsf{M}_{i,j} \leftarrow \mathsf{HMAC}(\Gamma.\mathsf{C}_{i,j}, 0\text{x}1)$	1 : $\Sigma.\mathsf{M}_{i,j} \leftarrow \mathsf{HMAC}(\Sigma.\mathsf{C}_{i,j}, 0\text{x}3)$

Com as novas message keys, o setup inicial da sessão pode ser compartilhado com segurança com cada membro da sala usando o esquema de criptografia em dupla camada baseado em criptografia simétrica padrão e desenvolvida pelo governo. No lado receptor, a mensagem criptografada contendo o novo setup inicial da sessão msgGX pode ser descriptografada ao avançar as root keys, chain keys e message keys correspondentes ❾. Uma vez descriptografada, a sessão msgGX pode ser reconstruída por cada destinatário, e a sala retoma a troca de mensagens usando a sessão atualizada.

3.5. Compatibilidade Retroativa

O msgX oferece suporte a um perfil de implantação compatível com Olm para interoperabilidade com clientes Matrix existentes no qual as etapas criptográficas desenvolvidas pelo governo são omitidas. No ramo padrão, o msgX-PQXDH (Figura 2) inclui um componente opcional $\Gamma.\mathsf{ss}$, que pode ser desabilitado para corresponder ao comportamento original do Olm baseado em X3DH. Analogamente, operações PQ envolvendo $\Gamma.\mathsf{ss}_i$ são omitidas em Γ.msgX-ratchet.advanceRootKey(·). Quando implementado de forma modular, o msgX pode emular o Olm ao desabilitar recursos PQ e desenvolvidos pelo governo, permitindo migração gradual em vez de uma transição obrigatória em data única.

3.6. Algoritmos Padrão

A Tabela 1 resume as primitivas criptográficas usadas nos ramos padrão e desenvolvido pelo governo do protocolo msgX. A arquitetura híbrida assegura agilidade criptográfica e aumenta a resiliência contra adversários clássicos e quânticos.

Tabela 1. Algoritmos usados no protocolo msgX.

	Criptografia padrão	Criptografia desenvolvida pelo governo
Simétrica	AES-256-CBC	Segurança equivalente a 256 bits
Curva elíptica (acordo de chaves)	Curve25519 ou Curve448	Segurança equivalente a 256 ou 448 bits
Curva elíptica (assinatura)	Ed25519 ou Ed448	Segurança equivalente a 256 ou 448 bits
PQC (encapsulamento de chaves)	ML-KEM ou Katana	KEM PQC com nível de segurança de 256 bits
HMAC	HMAC-SHA-256	HMAC-SHA-256
HKDF	Baseada em HMAC-SHA-256	Baseada em HMAC-SHA-256

4. Análise de Segurança

4.1. Contexto de Segurança Herdado

O Olm [Matrix.org 2024] fornece confidencialidade, autenticidade, FS e PCS para canais par a par, embora a PCS dependa da assinatura de pré-chaves que a especificação não impõe por padrão [Albrecht et al. 2024], deixando espaço para ataques de compartilhamento indevido de chaves e personificação. O Megolm estende o Olm para grupos com um ratchet simétrico unidirecional, enfraquecendo FS e PCS pois estados anteriores do ratchet podem persistir em armazenamento, e a renovação de sessão é necessária para restaurar a segurança futura após um comprometimento. No nível do ecossistema, análises anteriores [Albrecht et al. 2023, Li et al. 2023] mostram que um homeserver malicioso pode explorar seu controle sobre a associação às salas, as listas de dispositivos e o roteamento de eventos para comprometer a confiança dos clientes. Essas vulnerabilidades decorrem da própria arquitetura do ecossistema e também afetam o msgX. Ainda assim, sua camada criptográfica foi projetada para preservar a segurança mesmo diante de um servidor malicioso que atue de forma passiva.

4.2. Propriedades de Segurança do msgX

São considerados adversários capazes de interceptar ou modificar ciphertexts, homeservers potencialmente maliciosos, adversários com capacidade quântica que empreguem HNDL e adversários capazes de comprometer apenas um dos ramos criptográficos. Para mitigar essas ameaças, todas as chaves incluídas no bundle de Bob são assinadas e verificadas por Alice antes da execução do msgX-PQXDH. Além disso, a integridade dos dados é protegida por AEAD em ambos os ramos.

O msgX-ratchet fornece FS em dois níveis: cada chamada `advanceRootKey` descarta root keys e chain keys anteriores, tornando irrecuperáveis setups passados de sessões msgGX, e o ratchet simétrico do msgGX deriva uma chave única por mensagem. A PCS segue a mesma estrutura: `advanceRootKey` reinjeta entropia assimétrica a cada 2^N renovações via ECDH e, nas fronteiras de KEM, um encapsulamento PQ novo, permitindo recuperação após um ciclo completo. O intervalo de renovação R do msgGX então

controla quão rapidamente um estado de sessão de grupo comprometido é substituído, conforme detalhado na Seção 5.

O projeto de dois ramos fornece redundância contra ameaças ortogonais: o setup da sessão msgGX é criptografado independentemente sob Γ e Σ, de modo que um adversário deve comprometer ambos os ramos simultaneamente. Um futuro CRQC capaz de quebrar o ramo padrão não comprometeria o ramo Σ. Da mesma forma, uma eventual fraqueza nas primitivas soberanas não afetaria o ramo Γ. A resistência a HNDL decorre da mesma construção, pois o msgX-PQXDH vincula o segredo de sessão tanto a um componente ECDH clássico quanto a um componente KEM PQ em cada ramo. Por fim, o msgX exige a assinatura das pré-chaves, mitigando a fraqueza identificada no Olm [Albrecht et al. 2024]. Os ataques no nível do homeserver [Albrecht et al. 2023, Li et al. 2023], por sua vez, permanecem fora do escopo criptográfico e devem ser tratados por mecanismos de segurança da implantação. As propriedades de segurança acima dependem de hipóteses de composição que devem ser verificadas formalmente em trabalhos futuros.

5. Avaliação Experimental

5.1. Objetivo, Perfis e Configuração Experimental

A avaliação teve como objetivo quantificar o custo marginal dos ramos criptográficos do msgX em relação ao Olm. Para isso, foram considerados três perfis de execução. O primeiro perfil corresponde ao *Olm*, linha de base clássica com X3DH e Double Ratchet. O segundo perfil corresponde ao *msgX*-Γ, que isola o ramo padrão com msgX-PQXDH, ML-KEM-1024 e Curve25519. O terceiro perfil corresponde ao *msgX*-Σ, que isola o ramo governamental com KEM PQC de nível 256 bits em conjunto com Curve448. Cada execução isola um único perfil de protocolo, permitindo decomposição do custo por ramo sem confundir as sobrecargas dos dois ramos do msgX.

Os experimentos foram executados em dois dispositivos: um Samsung Galaxy S25 Ultra (Snapdragon 8 Elite, ARM64, Android 15, 12 GB RAM, Wi-Fi 5) e um Samsung Galaxy S23 (Snapdragon 8 Gen 2, ARM64, Android 15, 8 GB RAM, Wi-Fi 5), ambos conectados ao mesmo *homeserver* Synapse local com PostgreSQL 15. Os resultados quantificam a sobrecarga em rede local controlada, portanto, a latência de federação em escala de Internet constitui trabalho futuro. Synapse e banco de dados foram apagados antes de cada execução individual para eliminar contaminação de estado.

5.2. Plano Experimental e Métricas

O plano experimental organiza-se em doze cenários de *benchmark*. As métricas coletadas abrangem latência de *bootstrap*, latência média por mensagem em regime estacionário, *overhead* nas mensagens de renovação do *ratchet*, custo de rotação de sessão msgGX, tempo de distribuição de chave de grupo, latência de entrada de novo membro (*handshake* PQXDH), tamanho do *payload* de OTK e consulta de chaves.

Três fatores foram variados, cada um com 4 níveis: (i) fator R (intervalo de *rekeying*), com níveis $R \in \{16, 32, 64, 128\}$ mensagens, que controla de quantas em quantas mensagens uma nova sessão OLM PQ é negociada; (ii) fator N (comprimento de sessão), com níveis $N \in \{32, 64, 128, 256\}$ mensagens, que verifica se há degradação com sessões mais longas; e (iii) fator G (tamanho do grupo), com níveis $G \in \{8, 16, 32, 64\}$ membros,

que caracteriza a escalabilidade da distribuição de chaves Megolm. Cenários cujo resultado não depende de R, N ou G são executados com $R = 32$ fixo. A Tabela 2 detalha os doze cenários, que foram repetidos 30 vezes por dispositivo. O conjunto de dados totaliza 14.040 execuções: 78 cfg. $\times$ 3 protocolos $\times$ 30 repetições $\times$ 2 dispositivos.

Tabela 2. Plano experimental por fator e número de configurações por protocolo.

Fator	Cenários	Cfg.
Fixo (R=32)	*Bootstrap*; rotação msgGX; *payload* OTK; consulta de chaves; persistência; reposição de OTKs	6×1
R	Renovação do *ratchet*; *broadcast* de *re-key* em grupo	2×4
$R\times N$	Latência sustentada por mensagem	16
$R\times G$	Distribuição de chave de grupo; entrada de novo membro; saída de membro	3×16
Total		**78**

5.3. Metodologia Estatística e Hipóteses

Adotamos o seguinte protocolo de análise para cada comparação. (1) Normalidade: Shapiro-Wilk; (2) Duas amostras independentes: teste-t de Welch quando as duas amostras são compatíveis com normalidade; Mann-Whitney U caso contrário; (3) Três protocolos: ANOVA unidirecional quando a comparação é paramétrica; Kruskal-Wallis quando a normalidade é rejeitada; (4) Tendência com G: Spearman para verificar associação monotônica entre tamanho do grupo e latência.

Todos os resultados reportam média $\pm$ IC$_{95\%}$. Cada hipótese define explicitamente H_0, H_A, nível de significância $\alpha = 0{,}05$ e decisão baseada no valor-p: rejeita-se H_0 se $p < \alpha$; caso contrário, não há evidência suficiente para rejeitá-la. Foram definidas 7 hipóteses (H_1–H_7, $\alpha = 0{,}05$), organizadas em 2 grupos. O primeiro grupo (H_1–H_5) avalia o comportamento dos protocolos em ambos os dispositivos: H_1–H_3 comparam latências de *bootstrap* entre os 3 perfis (Mann-Whitney e Welch-t); H_4 avalia se há diferença global na latência sustentada entre os protocolos por meio do teste de Kruskal–Wallis; H_5 verifica ausência de custo de renovação do *ratchet* (pareado por execução). O segundo grupo (H_6–H_7) compara os 2 dispositivos: H_6 e H_7 testam se *bootstrap* e latência sustentada PQ diferem entre S23 e S25 Ultra (Welch-t/Mann-Whitney).

5.4. Custo de Estabelecimento e Latência Sustentada

A Figura 6 compara as latências de estabelecimento da sessão e de processamento sustentado das mensagens nos dois dispositivos, enquanto a Tabela 4 reúne as principais métricas de desempenho. O custo de estabelecimento do msgX concentra-se no *handshake* msgX-PQXDH. No S25 Ultra, a latência de *bootstrap* aumenta de 717 ± 20 ms com o Olm para 989 ± 36 ms com o msgX-Γ, correspondendo a um acréscimo de 38%. Para o msgX-Σ, a latência alcança 1095 ± 38 ms, um aumento de 53% em relação ao Olm. Ambas as diferenças são estatisticamente significativas (H_1 e H_2, $p < 10^{-10}$). A diferença entre os perfis Γ e Σ também é significativa (H_3, $p < 10^{-3}$), evidenciando a sobrecarga adicional do perfil governamental em relação ao baseado em ML-KEM-1024.

O mesmo comportamento é observado no S23, cujas latências de *bootstrap* são de 934 ± 28 ms para o Olm, 1286 ± 48 ms para o msgX-Γ e 1425 ± 57 ms para o msgX-Σ. Em comparação com o S25 Ultra, as diferenças são estatisticamente significativas para os três protocolos (H_6 rejeitada, $p < 0{,}001$), refletindo o maior custo de processamento no dispositivo menos recente.

Tabela 3. Hipóteses estatísticas avaliadas na Seção 5.

Hip.	Pergunta	H_0	H_A	Teste	Valor-p
H_1	*Bootstrap* msgX-Γ vs. Olm (ambos disp.)	$\mu_\Gamma = \mu_{\mathrm{Olm}}$	$\mu_\Gamma \neq \mu_{\mathrm{Olm}}$	Mann-Whitney	$< 0{,}001$
H_2	*Bootstrap* msgX-Σ vs. Olm (ambos disp.)	$\mu_\Sigma = \mu_{\mathrm{Olm}}$	$\mu_\Sigma \neq \mu_{\mathrm{Olm}}$	Mann-Whitney	$< 0{,}001$
H_3	*Bootstrap* msgX-Σ vs. msgX-Γ (ambos disp.)	$\mu_\Sigma = \mu_\Gamma$	$\mu_\Sigma \neq \mu_\Gamma$	Welch-t	$0{,}001$
H_4	Latência sustentada entre protocolos (ambos disp.)	$\mu_{\mathrm{Olm}} = \mu_\Gamma = \mu_\Sigma$	Pelo menos uma média difere	Kruskal-Wallis	$< 0{,}001$
H_5	Renovação do *ratchet* vs. mensagem normal	$\mu_{\mathrm{rekey}} = \mu_{\mathrm{normal}}$	$\mu_{\mathrm{rekey}} \neq \mu_{\mathrm{normal}}$	Pareado (t)	Olm: 0,29; Γ: 0,67; Σ: 0,02
H_6	*Bootstrap* S23 vs. S25 Ultra	$\mu_{\mathrm{S23}} = \mu_{\mathrm{S25U}}$	$\mu_{\mathrm{S23}} \neq \mu_{\mathrm{S25U}}$	Welch-t/Mann-Whitney	$< 0{,}001$
H_7	Sustentado PQ S23 vs. S25 Ultra	$\mu_{\mathrm{S23}} = \mu_{\mathrm{S25U}}$	$\mu_{\mathrm{S23}} \neq \mu_{\mathrm{S25U}}$	Welch-t/Mann-Whitney	$< 0{,}001$

μ representa a média populacional da métrica avaliada. A escolha paramétrica/não-paramétrica é determinada pelo teste de Shapiro-Wilk ($\alpha = 0{,}05$). H_5: não rejeitada para Olm e msgX-Γ; rejeitada para msgX-Σ ($+2{,}2\%$, $p = 0{,}02$), diferença de magnitude negligenciável na prática.

Tabela 4. Resumo de desempenho nos dois dispositivos (médias $\pm$ IC$_{95\%}$, $n = 30$). Métricas algorítmicas independem do dispositivo.

	Olm		msgX-Γ		msgX-Σ	
Métrica	S25 Ultra	S23	S25 Ultra	S23	S25 Ultra	S23
Bootstrap (ms)	717±20	934±28	989±36	1286±48	1095±38	1425±57
Sustentado (ms/msg)	444±2	421±6	651±2	705±10	794±9	853±12
Overhead ratchet (R=128)	≈0%		≈0%		≈0%	
Overhead rotação msgGX	+12,1%		+9,0%		+11,2%	
Payload OTK (B)	395		3.700		4.042	
Distrib. grupo G=64 (s)	21,1	22,1	34,3	37,0	39,7	42,6
PQXDH novo membro (ms)	489±5	525±1	787±8	852±2	942±24	1022±3

H_5 não rejeitada para Olm e Γ; overhead marginal ($+2{,}2\%$) para Σ. *Payload* OTK: $9{,}4\times$/$10{,}2\times$ o Olm (independe do dispositivo). PQXDH com novo membro: custo $O(1)$ em G (sem tendência monotônica detectada).

Após o estabelecimento da sessão, a latência por mensagem permanece estável para todos os níveis do fator $N \in \{32, 64, 128, 256\}$, sem evidência de degradação à medida que a sessão se prolonga. No S25 Ultra, foram observadas latências de 444 ± 2 ms/msg para o Olm, 651 ± 2 ms/msg para o msgX-Γ e 794 ± 9 ms/msg para o msgX-Σ. O teste global de Kruskal–Wallis indica diferenças estatisticamente significativas entre as três distribuições (H_4, $p < 10^{-10}$).

No S23, a latência sustentada do Olm é de 421 ± 6 ms/msg, valor próximo aos 444 ms/msg observados no S25 Ultra. Esse resultado é consistente com o predomínio da componente de rede no protocolo clássico. Como os dois dispositivos utilizam o mesmo servidor Synapse, as diferenças de processamento entre os respectivos SoCs permanecem dentro da variabilidade de rede observada.

Por outro lado, o maior peso das operações criptográficas torna mais perceptível a diferença entre os dispositivos nos protocolos PQ. O S23 apresenta latências de 705 ± 10 ms/msg para o msgX-Γ, um aumento de $8{,}2\%$ em relação ao S25 Ultra, e de 853 ± 12 ms/msg para o msgX-Σ, correspondente a um aumento de $7{,}4\%$. Embora estatis-

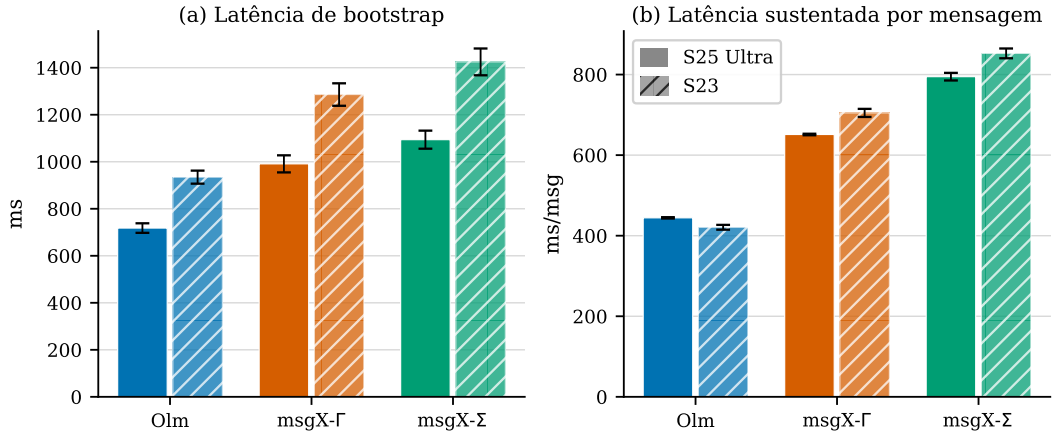


Figura 6. (a) Latência de *bootstrap* (descriptografia da 1ª mensagem, *handshake* PQXDH) e (b) latência sustentada por mensagem ($R = 32$), para os três protocolos em dois dispositivos. Barras sólidas: S25 Ultra; barras hachuradas: S23. Barras de erro: $\pm$ IC$_{95\%}$ ($n = 30$).

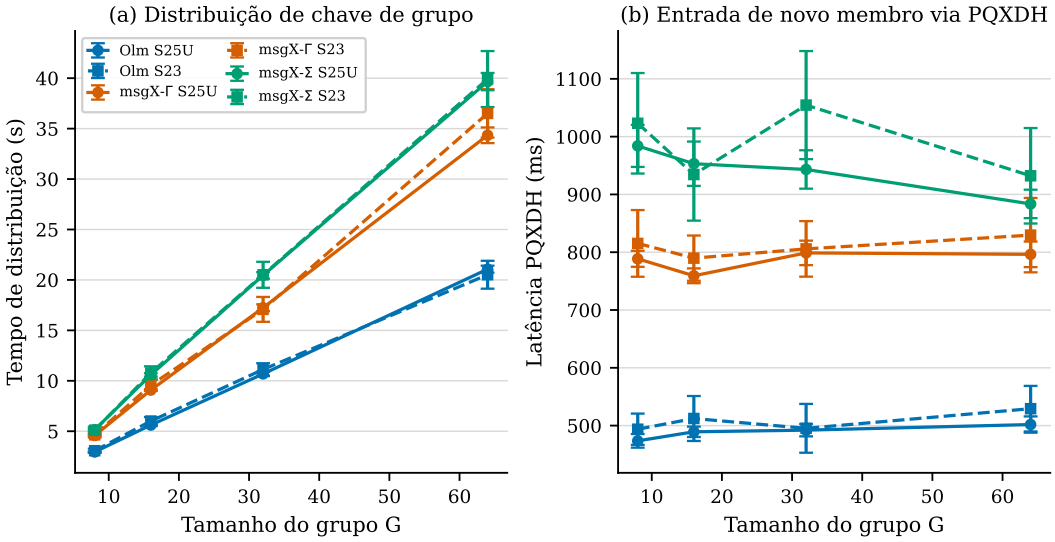


Figura 7. Escalabilidade de grupo (média e IC$_{95\%}$, $n = 30$). Linhas contínuas: S25 Ultra; tracejadas: S23. (a) Distribuição de chave cresce aproximadamente O(G) para todos os níveis do fator G. (b) PQXDH de novo membro permanece aproximadamente constante em G.

ticamente significativas (H_7, $p < 10^{-10}$), essas diferenças possuem magnitude moderada.

5.5. Overhead de Ratchet, Escalabilidade de Grupo e Payload OTK

A renovação do *ratchet* não apresenta custo estatisticamente significativo no S25 Ultra para o Olm ($p = 0{,}29$) nem para o msgX-Γ ($p = 0{,}67$), de modo que H_5 não é rejeitada nesses casos. Para o msgX-Σ, observa-se uma sobrecarga marginal de $+2{,}2\%$ ($p = 0{,}02$), equivalente a $\approx$17 ms por mensagem de renovação, valor pequeno diante da variabilidade de rede observada. A sobrecarga da rotação de sessão msgGX também permanece moderada, com $+12{,}1\%$ para o Olm, $+9{,}0\%$ para o msgX-Γ e $+11{,}2\%$ para o msgX-Σ, pois a rotação reutiliza a sessão existente sem realizar um novo *claim* de OTK.

Quanto à escalabilidade de grupo, a Figura 7 mostra que o tempo de distribuição de chaves cresce linearmente com os níveis do fator G em ambos os dispositivos, conforme esperado do modelo O(G) do Megolm. Para $G = 64$, os tempos são $21{,}1$ s / $22{,}1$ s para o Olm, $34{,}3$ s / $37{,}0$ s para o msgX-Γ e $39{,}7$ s / $42{,}6$ s para o msgX-Σ, sendo cada par correspondente ao S25 Ultra e ao S23, respectivamente.

Em contraste, o *handshake* PQXDH para a entrada de um novo membro não apresenta crescimento monotônico com o número de participantes existentes. Embora o teste de Kruskal–Wallis detecte variação entre os níveis de G, não se observa tendência de aumento. No msgX-Σ executado no S25 Ultra, por exemplo, a latência diminui de 979 ms em $G = 8$ para 885 ms em $G = 64$. Os valores médios são 489 ms / 525 ms para o Olm, 787 ms / 852 ms para o msgX-Γ e 942 ms / 1022 ms para o msgX-Σ. Assim, o msgX adiciona um custo PQ constante ao estabelecimento da sessão com o novo membro, preservando o comportamento assintótico O(G) do Megolm na redistribuição de chaves.

Por fim, o maior custo de largura de banda é o *payload* de OTK, que cresce de 395 B no Olm para 3.700 B no msgX-Γ ($9{,}4\times$) e 4.042 B no msgX-Σ ($10{,}2\times$). Tais valores são determinados pelos algoritmos e independem do dispositivo. Como a latência de *claim* permanece entre 18 e 22 ms, o principal impacto prático está no volume de material criptográfico armazenado e distribuído pelo *homeserver* quando há muitos usuários.

5.6. Síntese e Ameaças à Validade

Os resultados mostram que msgX-Γ e msgX-Σ introduzem sobrecarga constante em relação ao Olm, sem alterar as propriedades assintóticas herdadas do Megolm. Os principais custos concentram-se no *bootstrap* e no *payload* de OTK. A latência sustentada por mensagem, por sua vez, é predominantemente influenciada pela rede no protocolo clássico e apresenta diferenças moderadas entre os dispositivos nos protocolos PQ. Todavia, os experimentos realizados em uma rede local controlada suscitam avaliações adicionais para generalização a redes com maior latência.Além disso, os ramos Γ e Σ foram avaliados isoladamente, e o custo da configuração combinada $\Gamma+\Sigma$ não foi medido diretamente.

6. Conclusão

A ameaça de ataques HNDL torna urgente a transição pós-quântica em mensagens governamentais, enquanto a soberania digital exige independência de primitivas cujas futuras vulnerabilidades não possam ser auditadas nem substituídas pelo operador nacional. O msgX combina criptografia PQ padronizada e primitivas soberanas de forma interdependente: a quebra isolada de um ramo não revela texto claro, e o estabelecimento de sessão via msgX-PQXDH vincula o segredo a ambos os componentes de cada ramo, mitigando HNDL mesmo se um único vetor criptográfico for comprometido.

Ademais, o custo da proteção PQ soberana é viável em hardware Android atual e escalável para grupos de até 64 membros, oferecendo um caminho concretopara implantações nacionais do Matrix. Prioridades futuras incluem: (i) verificação formal das propriedades de composição dos dois ramos, requisito para adoção por órgãos de padronização; (ii) avaliação da configuração combinada $\Gamma+\Sigma$, que representa a implantação plena e não teve custo medido diretamente; (iii) experimentos em hardware restrito e redes de maior latência;e (iv) otimização do *payload* de OTK, cujo aumento de $9{,}4\times$/$10{,}2\times$ frente ao Olm é o principal impacto de largura de banda em servidores com muitos usuários.

Referências

[Albrecht et al. 2023] Albrecht, M. R., Celi, S., Dowling, B., and Jones, D. (2023). Practically-exploitable cryptographic vulnerabilities in matrix. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 164–181. IEEE. pages 9, 10

[Albrecht et al. 2024] Albrecht, M. R., Dowling, B., and Jones, D. (2024). Device-oriented group messaging: a formal cryptographic analysis of matrix'core. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE. pages 9, 10

[Auerbach et al. 2025] Auerbach, B., Dodis, Y., Jost, D., Katsumata, S., and Schmidt, R. (2025). How to compare bandwidth constrained two-party secure messaging protocols: a quest for a more efficient and secure post-quantum protocol. In *Proceedings of the 34th USENIX Conference on Security Symposium*, pages 6717–6736. pages 3

[Barnes et al. 2023] Barnes, R., Beurdouche, B., Robert, R., Millican, J., Omara, E., and Cohn-Gordon, K. (2023). The messaging layer security (mls) protocol. RFC 9420, Internet Engineering Task Force. pages 3

[Dodis et al. 2025] Dodis, Y., Jost, D., Katsumata, S., Prest, T., and Schmidt, R. (2025). Triple ratchet: A bandwidth efficient hybrid-secure signal protocol. In *Annual Int. Conference on the Theory and Applications of Cryptographic Techniques*, pages 302–331. Springer. pages 3

[Française 2019] Française, R. (2019). Tchap. `https://www.tchap.gouv.fr/`. pages 2

[Hodgson and Chathi 2023] Hodgson, M. and Chathi, H. (2023). A giant leap forwards for encryption with MLS. `https://matrix.org/blog/2023/07/a-giant-leap-with-mls/`. Published July 18, 2023. Accessed July 29, 2026. pages 3

[Kret and Schmidt 2023] Kret, E. and Schmidt, R. (2023). The pqxdh key agreement protocol. *`https://signal.org/docs/specifications/pqxdh/pqxdh.pdf`*. pages 2, 3

[Li et al. 2023] Li, H., Wu, Y., Huang, R., Mi, X., Hu, C., and Guo, S. (2023). Demystifying decentralized matrix communication network: Ecosystem and security. In *2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 260–267. IEEE. pages 9, 10

[Mahy and Barnes 2026] Mahy, R. and Barnes, R. L. (2026). Ml-kem and hybrid cipher suites for messaging layer security. Internet-Draft draft-ietf-mls-pq-ciphersuites-04, Internet Engineering Task Force. Work in Progress. pages 3

[Marlinspike and Perrin 2016] Marlinspike, M. and Perrin, T. (2016). The x3dh key agreement protocol. *Open Whisper Systems*, 283(10):10. pages 2

[Matrix.org 2019] Matrix.org (2019). Olm: A cryptographic ratchet. `https://gitlab.matrix.org/matrix-org/olm/-/blob/master/docs/olm.md`. libolm repository. pages 3

[Matrix.org 2022] Matrix.org (2022). Megolm group ratchet. `https://gitlab.matrix.org/matrix-org/olm/-/blob/master/docs/megolm.md`. libolm repository. pages 3

[Matrix.org 2024] Matrix.org (2024). Matrix specification. `https://spec.matrix.org/`. pages 2, 3, 9

[Matrix.org Foundation] Matrix.org Foundation. Vodozemac: An implementation of Olm and Megolm in pure Rust. `https://github.com/matrix-org/vodozemac`. Accessed July 29, 2026. pages 3

[Nist 2024] Nist, G. M. D. (2024). Module-lattice-based key-encapsulation mechanism standard. Technical report, National Institute of Standards and Technology, Gaithersburg, MD. pages 2

[Pacheco et al. 2022] Pacheco, R., Braga, D., Passos, I., Araújo, T., Lagrota, V., and Coutinho, M. (2022). libharpia: a new cryptographic library for brazilian elections. In *Anais do XXII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 250–263. SBC. pages 2

[Perrin and Marlinspike 2016] Perrin, T. and Marlinspike, M. (2016). The double ratchet algorithm. *GitHub wiki*, 112(4). pages 2, 7

[Schmidt and Connell 2025] Schmidt, R. and Connell, G. (2025). The ml-kem braid protocol. Signal Messenger Specification. Available at: `https://signal.org/docs/specifications/mlkembraid/`. pages 3

[Shor 1994] Shor, P. W. (1994). Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee. pages 2

Ecossistema de Comunicações Soberano Brasileiro

Vinícius Lagrota[1], Rodrigo Pacheco[1], Matheus Engel[1], Vanderson Rocha[1]

[1] Research and Development Center for Communication Security (CEPESC)
Brasília, DF – Brazil.

vinicius.lagrota@gmail.com, rppacheco.85@gmail.com,
matheus_spe@poli.ufrj.br, somrocha@gmail.com

Abstract. *Este artigo discute a soberania digital estatal como requisito estratégico para a proteção de dados, ativos estratégicos, infraestruturas críticas e comunicações sensíveis. Apresenta-se um ecossistema soberano de comunicações brasileiro, estruturado em torno de uma ICP soberana, bibliotecas criptográficas, sistemas operacionais hardenizados, aplicações seguras e plataformas de hardware nacionais. O ecossistema integra identidade digital, custódia de chaves, criptografia de Estado e mecanismos pós-quânticos em diferentes cenários de uso. O ecossistema busca reduzir dependências tecnológicas e jurisdicionais externas, fortalecendo a autonomia, a segurança e o controle institucional do Estado brasileiro sobre ativos estratégicos de informação.*

1. Introdução

A transformação digital ampliou a dependência do Estado de infraestruturas computacionais e canais eletrônicos para conduzir políticas públicas, custodiar informações sensíveis e coordenar funções críticas de governo. Nesse cenário, comunicações seguras tornam-se essenciais para garantir a capacidade estatal de preservar confidencialidade, integridade, disponibilidade e autenticidade em processos administrativos, diplomáticos, militares e de inteligência.

Essa dependência ocorre em um ambiente marcado por cadeias tecnológicas globais, concentração de plataformas e assimetrias jurisdicionais. Quando aplicações, nuvens, componentes de software ou mecanismos criptográficos estratégicos permanecem subordinados a provedores, padrões ou marcos legais estrangeiros, o Estado se expõe a riscos de dependência, descontinuidade, indisponibilidade e acesso extraterritorial a dados. Assim, a dimensão tecnológica passa a integrar diretamente a agenda de soberania estatal [União Europeia 2016, Congressional Research Service 2018, Commission Nationale de l'Informatique et des Libertés 2024].

A evolução da computação quântica intensifica esse desafio. Os algoritmos de Shor e Grover indicam que parte das bases criptográficas atualmente empregadas pode ser afetada por avanços nesse campo [Shor 1997, Grover 1996]. Em resposta, países vêm estruturando estratégias de transição para criptografia pós-quântica e adaptação de infraestruturas críticas, como mostram publicações do *National Institute of Standards and Technology* (NIST), do *National Cyber Security Centre* (NCSC) e da *Agence Nationale de la Sécurité des Systèmes d'Information* (ANSSI) [Moody et al. 2024, NCS 2025, ANS 2023, Barker et al. 2025]. Para o Estado brasileiro, esse movimento reforça a necessidade de combinar segurança criptográfica, autonomia tecnológica e controle institucional sobre plataformas de comunicação.

Este artigo tem dois objetivos principais: i) discutir a soberania digital estatal, com ênfase na atuação de nações estrangeiras sobre dados, infraestruturas e serviços críticos, e ii) apresentar o ecossistema soberano de comunicações brasileiro, concebido para elevar o controle nacional sobre identidades digitais, bibliotecas criptográficas, sistemas e plataformas de hardware. O trabalho situa o ecossistema proposto como resposta técnica e institucional a um problema estratégico mais amplo.

O restante do artigo está organizado da seguinte forma. A Seção 2 discute soberania digital estatal, seus níveis, exemplos internacionais e o contexto brasileiro. A Seção 3 apresenta o ecossistema soberano de comunicações e seus principais componentes. A Seção 4 aborda seu funcionamento. Por fim, a Seção 5 apresenta as conclusões.

2. Soberania Digital

Soberania digital pode ser entendida como a capacidade de um Estado tomar decisões autônomas sobre sua infraestrutura digital e sua implementação tecnológica em território nacional [Pohle and Thiel 2020]. Mais do que posse física de ativos, envolve escolhas sobre arquiteturas, dependências externas, tratamento de dados e supervisão institucional, sob a ótica do interesse público. Trata-se de atributo graduado, observável em níveis como software, infraestrutura de processamento e armazenamento, comunicações, hardware, cadeia de suprimentos, governança regulatória e controle sobre dados estratégicos.

Sob essa perspectiva, soberania digital depende de autonomia tecnológica e controle sobre infraestrutura e dados, sem implicar autarquia. Requer capacidade de compreender, auditar, adaptar, substituir e sustentar tecnologias sensíveis, por código aberto com domínio local ou soluções proprietárias sob controle nacional. Também envolvem-se local de processamento, jurisdição do provedor, custódia de chaves criptográficas e risco de acesso extraterritorial a dados. Na Europa, a Regulação Geral de Proteção de Dados — *General Data Protection Regulation* (GDPR) reforçou o controle sobre dados pessoais [União Europeia 2016], enquanto o debate sobre nuvem evidenciou riscos associados ao *Clarifying Lawful Overseas Use of Data Act* (CLOUD Act) e impulsionou arranjos com maior controle público sobre serviços críticos [Congressional Research Service 2018, Commission Nationale de l'Informatique et des Libertés 2024].

Experiências internacionais ilustram essa agenda: na França, o Tchap, operado pela *Direction Interministérielle du Numérique* (DINUM) em infraestrutura estatal, integra uma estratégia de soberania em áreas sensíveis [Tchap / DINUM 2026]; na China, plataformas nacionais, regulação de aplicativos, segurança de dados e controle de transferências internacionais coordenam capacidade tecnológica doméstica, regulação e controle sobre ativos críticos de informação [State Council of the People's Republic of China 2021, State Council of the People's Republic of China 2024].

No Brasil, o tema combina proteção de dados, defesa e infraestrutura. A Lei Geral de Proteção de Dados Pessoais (LGPD) tutela dados pessoais como matéria de interesse nacional [Brasil 2018]; a Lei nº 12.598/2012 reconhece capacidades produtivas e tecnológicas nacionais em setores estratégicos e insere as Empresas Estratégicas de Defesa como instrumentos de preservação de competências sensíveis ao Estado [Brasil 2012]; e iniciativas como a Nuvem de Governo, gerida por órgãos ou empresas públicas como Serpro e Dataprev, expressam a busca por infraestrutura sob controle nacional [Serviço Federal de Processamento de Dados 2026]. Assim, no contexto brasileiro, soberania digital es-

tatal combina governança regulatória, domínio de infraestruturas críticas, fortalecimento de empresas e cadeias tecnológicas estratégicas e operação de serviços essenciais sob controle jurisdicional e técnico próprio, enquadramento no qual se insere o ecossistema soberano de comunicações.

3. Ecossistema Soberano de Comunicações

O ecossistema soberano de comunicações, proposto pelo Centro de Pesquisa e Desenvolvimento para a Segurança das Comunicações (CEPESC), articula uma Infraestrutura de Chaves Públicas (ICP) soberana, bibliotecas criptográficas próprias, componentes de software de desenvolvimento nacional e plataformas de hardware concebidas no país, com o objetivo de prover identidade digital, proteção de dados e canais seguros de comunicação sob controle nacional. Em vez de tratar esses elementos de forma isolada, o ecossistema os organiza como uma arquitetura integrada, na qual a confiança se estabelece na ICP soberana, as primitivas criptográficas são encapsuladas em bibliotecas especializadas e os serviços finais são disponibilizados por sistemas e dispositivos voltados a diferentes cenários de uso.

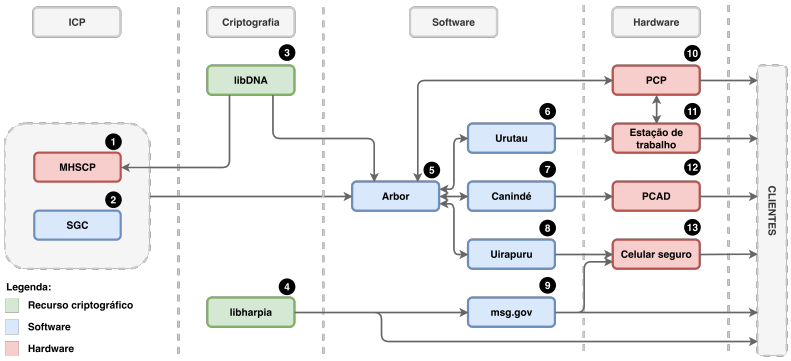


Figura 1. Visão geral do ecossistema soberano de comunicações e de sua infraestrutura criptográfica.

A Figura 1 sintetiza essa arquitetura em quatro domínios complementares: ICP, criptografia, software e hardware. Na Subseção 3.1, são detalhados os componentes da ICP soberana, representados pelos itens ❶ e ❷; na Subseção 3.2, são apresentadas as bibliotecas criptográficas correspondentes aos itens ❸ e ❹; na Subseção 3.3, são descritas as soluções de software associadas aos itens ❺ a ❾; e na Subseção 3.4, são discutidas as plataformas de hardware indicadas pelos itens ❿ a ⓭ da Figura 1.

3.1. Infraestrutura de Chaves Públicas soberana

Na arquitetura sintetizada pela Figura 1, a ICP soberana[1] é composta pelo Módulo de Hardware Seguro Criptográfico Proprietário (MHSCP) ❶ e pelo Sistema Gerenciador de Certificados Digitais (SGC) ❷. Em conjunto, esses componentes sustentam a raiz de confiança do ecossistema, viabilizando a custódia de material criptográfico sensível e o gerenciamento operacional do ciclo de vida dos certificados.

MHSCP ❶. É um Módulo de Segurança de Hardware — *Hardware Security Module* (HSM) adaptado para suportar, além da criptografia de mercado, mecanismos de

[1] A ICP-Brasil já é uma ICP soberana e pode ser perfeitamente adaptada para se aplicar ao contexto deste trabalho.

criptografia de Estado. É nesse equipamento que se concentra a raiz de confiança do ecossistema; por isso, sua implantação deve observar requisitos rigorosos para geração, armazenamento e uso da chave da Autoridade Certificadora (AC) raiz e das demais ACs subordinadas.

SGC ❷. É responsável por administrar as chaves criptográficas custodiadas no MHSCP e por operacionalizar os fluxos associados à certificação digital. Além disso, disponibiliza uma Interface de Programação de Aplicações — *Application Programming Interface* (API) por meio da qual o Arbor (ver Seção 3.3) envia requisições e recebe certificados emitidos pelas ACs.

3.2. Bibliotecas criptográficas

No domínio criptográfico representado na Figura 1, destacam-se a libDNA ❸ e a libharpia [Pacheco et al. 2022] ❹. Essas bibliotecas concentram a implementação das primitivas criptográficas e expõem interfaces que permitem sua incorporação segura por sistemas de mais alto nível.

libDNA ❸. É uma biblioteca compatível com PKCS #11 que pode atuar, por si só, como um *soft token*, permitindo o armazenamento seguro de chaves. Em repouso, seus dados permanecem cifrados com proteção derivada do PIN individual do usuário. Por meio da interface PKCS #11, outras aplicações também podem consumir os serviços criptográficos oferecidos pela biblioteca.

libharpia ❹. Fornece uma API de mais alto nível, projetada para reduzir a necessidade de parametrizações manuais por parte do usuário. Seu objetivo é favorecer o emprego consistente de mecanismos criptográficos, evitando que decisões sensíveis, como a escolha de algoritmos e modos de operação, sejam delegadas a aplicações sem o devido controle.

3.3. Soluções em software

Na camada de software mostrada na Figura 1, encontram-se o Arbor ❺, o Urutau ❻, o Canindé ❼, o Uirapuru ❽ e o msg.gov ❾. Esses componentes fazem a mediação entre a infraestrutura criptográfica e os ambientes de uso, distribuindo capacidades de identidade, proteção de dados e comunicação segura de acordo com o perfil de cada plataforma.

Arbor ❺. Ocupa posição central nessa camada. Sua função é orquestrar o acesso às bibliotecas criptográficas e à ICP soberana, oferecendo uma base comum para os demais sistemas. Entre seus principais módulos, destacam-se: i) Cellulam: *wrapper* implementado em Go para a libDNA, com o propósito de simplificar sua integração com outros sistemas; ii) Cítala: camada construída sobre o Cellulam que expõe os recursos criptográficos para aplicações consumidoras e trata aspectos de concorrência; e iii) Suber: módulo responsável pela comunicação entre o Arbor e a ICP soberana.

Urutau ❻. É um Sistema Operacional (SO) hardenizado, baseado na distribuição Debian, concebido para reduzir a superfície de ataque e oferecer um ambiente operacional mais controlado. Ao incorporar as funcionalidades criptográficas providas pelo Arbor, o sistema viabiliza, entre outras capacidades, conexões por Rede Privada Virtual — *Virtual Private Network* (VPN) para clientes com suporte tanto à criptografia de Estado quanto à criptografia de mercado.

Canindé ❼. Deriva do Urutau e preserva seus mecanismos de endurecimento, mas é direcionado a equipamentos que operam, principalmente, como servidores VPNs. Por essa razão, contempla adaptações voltadas a cenários de maior demanda de processamento e tráfego de rede.

Uirapuru ❽. Também é um SO hardenizado baseado em código aberto, mas voltado ao ecossistema Android. Além da redução da superfície de ataque, sua proposta inclui diminuir a dependência de serviços providos por grandes plataformas digitais, ampliando o controle sobre os dados e a privacidade do usuário.

msg.gov ❾. É um aplicativo de mensageria segura que emprega os recursos criptográficos disponibilizados pela libharpia e se apoia no ecossistema Matrix [Martins et al. 2026], cuja arquitetura federada permite a interoperabilidade entre *homeservers* administrados de forma independente. O sistema utiliza os protocolos msgX e msgGX, derivados de Olm e Megolm[2] [Lagrota et al. 2025], respectivamente, com adaptações destinadas a introduzir Criptografia Pós-Quântica — *Post-Quantum Cryptography* (PQC) tanto na negociação quanto na derivação de chaves. O desenho do protocolo prevê uma dupla camada criptográfica: uma baseada em primitivas de mercado e outra em primitivas de Estado.

3.4. Soluções em hardware

No domínio de hardware da Figura 1, situam-se a Plataforma Criptográfica Portátil (PCP) ❿, a estação de trabalho ⓫, a Plataforma Criptográfica de Alto Desempenho (PCAD) ⓬ e o celular seguro ⓭. Esses elementos materializam, em plataformas específicas, as capacidades criptográficas, de identidade digital e de comunicação segura providas pelo ecossistema.

PCP ❿. É uma plataforma portátil que embarca, por meio do Arbor, a libDNA, expondo capacidades criptográficas para cifração e decifração de dados, bem como permitindo o armazenamento perene de chaves, o que faz dela o *token* físico da identidade digital do usuário. Sua autenticidade é fornecida pela ICP soberana, com a qual se integra via Arbor. O dispositivo também dispõe de armazenamento cifrado para transporte seguro de dados e incorpora o protocolo FIDO2 para autenticação.

Estação de trabalho ⓫. Utiliza o SO Urutau para oferecer funcionalidades de segurança sem exigir hardware especializado. Quando associada à PCP, viabiliza autenticação via FIDO2, cifração de dados e operação como cliente VPN, usando a PCP como motor criptográfico em hardware.

PCAD ⓬. Destina-se principalmente a conexões VPN de alto desempenho, com emprego típico como servidor em arquiteturas em estrela, para criptografar todo o tráfego de uma rede. A plataforma embarca o SO Canindé e obtém suas capacidades criptográficas da libDNA por meio do Arbor, com recursos nativos ou apoio de módulos dedicados baseados em Matriz de Portas Programável em Campo — *Field-Programmable Gate Array* (FPGA) ou Processador Digital de Sinais — *Digital Signal Processor* (DSP). Também é via Arbor que a PCAD encaminha à ICP soberana requisições de certificação digital.

Celular seguro ⓭. Serve de base para o SO Uirapuru, ampliando a proteção em relação

[2]Os protocolos Olm e Megolm são baseados no protocolo Signal [Matrix.org 2019, Matrix.org 2022].

a sistemas convencionais. Integrado ao Arbor, pode prover identidade digital. Quando combinado ao msg.gov, protege dados em repouso por meio do sistema operacional e assegura confidencialidade e integridade dos dados em trânsito por meio da aplicação.

4. Funcionamento do Ecossistema Soberano de Comunicações

A Figura 2 sintetiza formas típicas de emprego do ecossistema soberano de comunicações. Em todos os cenários, a confiança deriva da ICP soberana, enquanto o Arbor medeia aplicações e bibliotecas criptográficas, viabilizando identidade digital, cifração e canais seguros em diferentes plataformas.

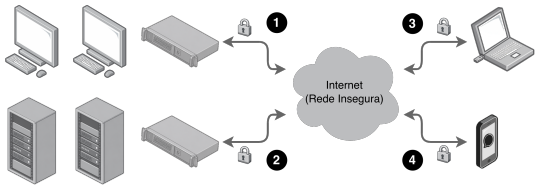


Figura 2. Diagrama do funcionamento do Ecossistema Soberano de Comunicações.

Nos cenários ❶ e ❷, a PCAD atua, respectivamente, como cliente VPN em um escritório remoto e como servidor VPN em um Central de Processamento de Dados (CPD). No primeiro caso, usuários de uma rede geograficamente distante acessam, por túneis criptográficos, serviços e aplicações da sede da organização; no segundo, a PCAD concentra múltiplos túneis e controla o ingresso desse tráfego na infraestrutura protegida. No cenário ❸, uma estação de trabalho com o SO Urutau opera com a PCP, combinando endurecimento do sistema, autenticação forte e proteção do material criptográfico no lado do usuário, especialmente em ambientes externos sem garantia de segurança cibernética, como reuniões e missões diplomáticas no exterior.

O cenário ❹ representa o uso do celular seguro com o SO Uirapuru e o msg.gov, estendendo o ecossistema à mobilidade e à comunicação segura. O dispositivo protege dados em repouso, enquanto o aplicativo assegura confidencialidade e integridade das mensagens em trânsito. Em conjunto, os quatro casos mostram que o ecossistema atende ao acesso seguro a redes e serviços e à comunicação protegida entre usuários, preservando controle nacional sobre identidades, chaves e plataformas.

5. Conclusão

Este trabalho discutiu a soberania digital como problema tecnológico, regulatório e estratégico, destacando que a dependência de infraestruturas, plataformas e marcos jurisdicionais estrangeiros amplia riscos à proteção de dados, à continuidade operacional e à autonomia decisória do Estado. Nesse contexto, apresentou-se o ecossistema soberano de comunicações brasileiro como arquitetura integrada para prover identidade digital, custódia de chaves, serviços criptográficos e canais seguros sob maior controle nacional. Ao articular uma ICP soberana, bibliotecas criptográficas, sistemas operacionais hardenizados, aplicações e plataformas de hardware especializadas, o ecossistema materializa a soberania digital no acesso seguro a redes e infraestruturas críticas e na comunicação protegida em ambientes móveis. Assim, a proposta fortalece a capacidade estatal ao alinhar segurança, autonomia tecnológica e controle institucional sobre ativos estratégicos de informação.

Referências

[ANS 2023] (2023). Anssi views on the post-quantum cryptography transition. Technical report, French Cybersecurity Agency (ANSSI). pages 1

[NCS 2025] (2025). Timelines for migration to post-quantum cryptography. Technical report, National Cyber Security Centre (NCSC). pages 1

[Barker et al. 2025] Barker, E., Chen, L., Moody, D., Regenscheid, A., Souppaya, M., Newhouse, B., Housley, R., and Turner, S. (2025). Considerations for achieving crypto agility: Strategies and practices. NIST Cybersecurity White Paper (CSWP) CSWP 39 (2nd Public Draft), National Institute of Standards and Technology (NIST), Gaithersburg, MD. pages 1

[Brasil 2012] Brasil (2012). Lei no 12.598, de 21 de março de 2012. Acesso em: 9 maio 2026. pages 2

[Brasil 2018] Brasil (2018). Lei no 13.709, de 14 de agosto de 2018: Lei geral de proteção de dados pessoais (lgpd). Acesso em: 9 maio 2026. pages 2

[Commission Nationale de l'Informatique et des Libertés 2024] Commission Nationale de l'Informatique et des Libertés (2024). Cloud: the risks of a european certification allowing foreign authorities access to sensitive data. Acesso em: 9 maio 2026. pages 1, 2

[Congressional Research Service 2018] Congressional Research Service (2018). Law enforcement access to overseas data under the cloud act. Acesso em: 9 maio 2026. pages 1, 2

[Grover 1996] Grover, L. K. (1996). A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. pages 1

[Lagrota et al. 2025] Lagrota, V., Maia, G., Rego, P., Pacheco, R., Macêdo, J. A., and Dantas, M. (2025). msgx: a national digital sovereignty matter. In *Simpósio Brasileiro de Cibersegurança (SBSeg)*, pages 344–345. SBC. pages 5

[Martins et al. 2026] Martins, J. A. P., Rego, P. A. L., de Macêdo, J. A. F., et al. (2026). Matrix protocol: a comprehensive systematic mapping study. *Journal of Cloud Computing*, 15(1):20. pages 5

[Matrix.org 2019] Matrix.org (2019). Olm: A cryptographic ratchet. `https://gitlab.matrix.org/matrix-org/olm/-/blob/master/docs/olm.md`. libolm repository. pages 5

[Matrix.org 2022] Matrix.org (2022). Megolm group ratchet. `https://gitlab.matrix.org/matrix-org/olm/-/blob/master/docs/megolm.md`. libolm repository. pages 5

[Moody et al. 2024] Moody, D., Perlner, R., Regenscheid, A., Robinson, A., and Cooper, D. (2024). Transition to post-quantum cryptography standards. Technical report, National Institute of Standards and Technology (NIST). pages 1

[Pacheco et al. 2022] Pacheco, R., Braga, D., Passos, I., Araújo, T., Lagrota, V., and Coutinho, M. (2022). libharpia: a new cryptographic library for brazilian elections. In

Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg), pages 250–263. SBC. pages 4

[Pohle and Thiel 2020] Pohle, J. and Thiel, T. (2020). Digital sovereignty. *Pohle, J. & Thiel*. pages 2

[Serviço Federal de Processamento de Dados 2026] Serviço Federal de Processamento de Dados (2026). Nuvem de governo. Acesso em: 24 maio 2026. pages 2

[Shor 1997] Shor, P. W. (1997). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509. pages 1

[State Council of the People's Republic of China 2021] State Council of the People's Republic of China (2021). China regulates apps' collection of personal information. Acesso em: 9 maio 2026. pages 2

[State Council of the People's Republic of China 2024] State Council of the People's Republic of China (2024). China issues regulations on network data security management. Acesso em: 9 maio 2026. pages 2

[Tchap / DINUM 2026] Tchap / DINUM (2026). À propos de tchap. Acesso em: 9 maio 2026. pages 2

[União Europeia 2016] União Europeia (2016). Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 (general data protection regulation). Acesso em: 9 maio 2026. pages 1, 2

Journal of Cloud Computing

REVIEW Open Access

Matrix protocol: a comprehensive systematic mapping study

José A. P. Martins[1], Paulo A. L. Rego[1], José A. F. de Macêdo[1], Francisco Airton Silva[2*] and Vinícius Lagrota[3]

Abstract
The Matrix protocol is gaining significant traction in the domain of decentralized communication, offering robust features like end-to-end encryption, federated server architecture, and seamless interoperability. As more applications and platforms adopt this protocol, understanding its performance and security implications becomes increasingly crucial. Evaluating the Matrix protocol's adoption and its impact on communication systems is challenging due to its decentralized nature, diverse implementation scenarios, and varying security requirements. Existing studies provide fragmented insights, lacking a comprehensive overview of how the Matrix protocol performs in real-world settings, especially concerning security, scalability, and interoperability. This paper presents a systematic mapping study, which rigorously examines 156 papers, narrowing them down to 21 primary studies that directly engage with the Matrix protocol. The study categorizes the research based on the relationship with the Matrix protocol, the main area of focus, the contributions made, and the specific features of the Matrix protocol explored. By employing visualizations and detailed cross-analysis, the study reveals key trends, research gaps, and provides a foundational understanding of the protocol's strengths and limitations in decentralized communication. This study aims to guide future research efforts, advocating for more structured evaluation frameworks to enhance the understanding and deployment of the Matrix protocol in diverse communication environments.

Keywords Systematic mapping study, Matrix protocol, Matrix basic concepts, Area of study, Research contributions

Introduction

Context and motivation

According to the Matrix.org Foundation, as a concept, *"the Matrix is an open protocol for decentralized, secure communications, developed to enable interoperability between different communication systems"* [1]. As digital communication continues to evolve, the importance of secure and private interactions has never been more significant [2]. Matrix offers a robust framework that prioritizes user control and data sovereignty, allowing individuals and organizations to maintain ownership of their communications. This is especially important in a world where centralized platforms often pose risks related to data breaches and unauthorized surveillance. The protocol's unique ability to provide secure, real-time communication across a federated network of independent servers makes it highly adaptable to the needs of modern distributed systems. This adaptability is further enhanced by Matrix's support for a wide range of communication types, including instant messaging, voice-over-IP (VoIP), and even video conferencing. Unlike traditional centralized messaging platforms, Matrix enables interoperability between different services, allowing users to communicate seamlessly across various networks [3]. This interoperability is a key differentiator

*Correspondence:
Francisco Airton Silva
faps@ufpi.edu.br
[1]Federal University of Ceará (UFC), Fortaleza, Ceará, Brazil
[2]Federal University of Piauí (UFPI), Picos, Piauí, Brazil
[3]Research and Development Center for Communication Security (CEPESC), Brasília, Distrito Federal, Brazil

© The Author(s) 2025. **Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

that positions Matrix as a forward-thinking solution in the landscape of digital communication. By breaking down the silos typically created by proprietary messaging systems, Matrix fosters a more connected and inclusive communication environment, where users are not restricted by the boundaries of specific platforms.

As a result of these strengths, the popularity of the Matrix protocol is rapidly increasing, with its adoption spreading across a variety of applications, from secure messaging and VoIP services to decentralized collaboration tools [4]. This widespread adoption is not only a testament to the protocol's technical capabilities but also reflects a growing awareness of the importance of privacy and security in digital communication. More organizations and communities are recognizing the need to replace from centralized systems—that can be vulnerable to breaches and data leaks—by the Matrix due to its ability to offer more security and resiliency [5]. The versatility of the Matrix protocol is another key factor driving its popularity. It is not limited to just one type of application or use case but can be adapted to meet a wide range of communication needs. The Matrix protocol has been widely adopted by governments, companies, and organizations due to its ability to provide decentralized, secure, and interoperable communication. A notable example is the French government, which developed Tchap, a platform based on Matrix, to ensure digital sovereignty and security in communications between public officials [6]. Another case is the NHS in the United Kingdom, which uses Matrix to establish a secure communication network compliant with privacy regulations such as the GDPR in its medical operations [7]. Cities like Berlin have also adopted the protocol to improve the efficiency and resilience of governmental communications [8].

Research problem

A federated architecture, like Matrix, enables decentralized and interconnected server operations. In this model, each server maintains autonomy while exchanging information with other network servers, creating a distributed system without a central control point. Combined with the necessity for interoperability across various platforms, makes it difficult to perform comprehensive evaluations that can guide the development and optimization of Matrix-based systems [9]. Existing research often provides fragmented insights, lacking a unified framework for assessing the protocol's performance under different conditions and usage scenarios. As organizations increasingly adopt Matrix for critical communication needs, the absence of a structured approach to evaluate its effectiveness and security hinders the ability to fully capitalize on its potential. The lack of standardized metrics and methodologies for assessing Matrix's performance in real-world deployments further complicates efforts to optimize its usage. This problem underscores the urgent need for a systematic mapping study that consolidates existing knowledge, identifies the critical aspects of Matrix evaluation, and provides a clear direction for future research. By addressing these challenges, this study aims to establish a foundation that can support more effective and informed use of the Matrix protocol in a wide range of applications.

Contribution

To address these challenges, this paper presents a comprehensive and focused systematic mapping study dedicated exclusively to the Matrix protocol—an area that has not yet been extensively explored in the literature. Rather than merely listing existing works, our study provides a structured classification and critical analysis of how the protocol has been adopted, evaluated, and discussed in academic contexts. The main contributions of this study include:

- A thorough identification and categorization of 156 papers, leading to the selection of 21 primary studies that directly engage with the Matrix protocol;
- A novel classification framework that maps each study in terms of its relationship with the protocol, research area, key contributions, and explored features;
- A detailed cross-analysis that reveals prevailing research trends, underexplored areas, and critical challenges that hinder the broader adoption and evaluation of Matrix;
- The provision of visual and tabular syntheses to support the understanding of the current research landscape and facilitate future comparative studies;
- An evidence-based discussion that informs and guides researchers and practitioners about directions for future work on decentralized, secure communication systems.

Outline

The remainder of this paper is organized as follows: Sect. 2 presents the Matrix protocol in details. Section 3 reviews related work, providing context and highlighting key contributions in the field. Section 4 details the methodology used in our systematic mapping study, outlining the steps taken to identify and analyze relevant literature. Section 5 presents the findings based on individual classifications, while Sect. 6 identifies the challenges encountered during the study and suggests future research directions. Finally, Sect. 7 concludes the paper with recommendations for further investigations in this area.

Overview of the Matrix protocol

Matrix protocol, an open standard for secure and decentralized communication, is gaining significant attention from academia, industry, and governments. Matrix builds upon existing components to establish an open protocol that empowers users to have full control over their data and communication. It is important to explore the historical context that influenced its development to understand its impact and usefulness. Additionally, we have examined the architecture that supports its distributed nature and the various use cases and applications that have emerged across different sectors. This section covers the evolution of the Matrix protocol and examines its technical framework. It also highlights the wide range of implementations demonstrating flexibility and adaptability in today's communication landscape.

Historical context

Matthew Hodgson and Amandine Le Pape are the cofounders of Matrix. Both initially worked at Amdocs, where they conceived Matrix to provide a more open and secure framework for real-time communication. The central idea was to create a decentralized communication infrastructure enabling interoperability between different messaging services. This approach aims to overcome the limitations of centralized protocols that create "communication silos" [1].

Table 1 summarizes the major changes in the Matrix protocol from its inception to the present year. It highlights significant milestones in the protocol's evolution, including version releases, security enhancements, adoption by governmental institutions, and the development of new features.

In May 2014, the team launched the first version of the Matrix protocol and the Riot client (later renamed Element), marking a significant step forward in decentralized communication. This initial release focused on interoperability and creating an open and decentralized communication network [6]. In July 2017, the Matrix.org Foundation was established as a non-profit organization to maintain and promote the development of the protocol as an open and free standard [1]. Later that year, E2EE was introduced into the protocol in October, significantly enhancing communications security on the Matrix network [1]. The year 2018 brought further innovations with the announcement of the beta version of the RiotX client in November. This was a complete rewrite of the Riot client, to provide a more modern and efficient user experience [6]. In June 2019, the Matrix protocol reached a significant milestone with the release of version 1.0 of its specification. This release solidified Matrix as a stable, production-ready solution, introducing security improvements, including detailed specifications for end-to-end encryption. Additionally, it formalized the protocol's APIs, such as the Client-Server API, Server-Server API, and Application Service API, establishing a clear standard for developers and integrators [1]. In December 2019, the RiotX client officially replaced the original Riot client, offering a faster and more secure experience [6].

In 2020, the protocol became more popular and adopted by various governmental institutions. This was especially notable in France, where Matrix was used to create "Tchap". Tchap is a secure communication system specifically designed for public officials [10]. In July of the same year, the Riot client was renamed to Element, consolidating its brand and market presence as the official client for the Matrix protocol [6]. Development continued in 2021 with the release of version 1.1 of the Matrix specification in February. This version introduced improvements in identity management and enhanced the robustness of server federation [1]. In October, the Spaces feature was introduced, allowing rooms to be organized into hierarchical groups, facilitating community management within Matrix [1]. In 2022, new versions of the protocol were released: version 1.2 in March, which brought optimizations in event synchronization and encryption enhancements [1], while version 1.3 in December focused on performance improvements and expanding federation capabilities [1]. The year 2023 saw significant updates. In May, version 1.4 was presented, focusing on greater compatibility with mobile devices and new security features [1]. In August, version 1.5 was released, bringing updates to the identity services API and support for new IoT (Internet of Things) use cases

Table 1 Chronological summary of changes in the Matrix protocol

Year	Historic Events
2014	Initial launch of the Matrix protocol and the first client, Riot (later renamed Element).
2017	Creation of the Matrix.org Foundation and introduction of end-to-end encryption (E2EE).
2018	Announcement of the beta version of RiotX, a major rewrite of the Riot client.
2019	Release of version 1.0 of the Matrix protocol specification and replacement of Riot with RiotX.
2020	Widespread adoption by governmental institutions, especially in France with the "Tchap" system. Riot renamed Element.
2021	Release of version 1.1 of the Matrix specification with improvements in identity management and server federation. Spaces feature introduced.
2022	Release of versions 1.2 and 1.3, focusing on event synchronization, encryption enhancements, and performance improvements.
2023	Releases 1.4–1.9, including compatibility improvements, moderation tools, and enhanced interoperability.
2024	Release of version 1.10–1.11 with security enhancements and refined user experience.

[1]. The protocol continued its evolution with the release of version 1.6 in June, which included enhanced support for AR and VR applications [1]. In August, version 1.7 improved interoperability with other decentralized communication networks, further strengthening Matrix's position as a versatile and secure solution for decentralized communication [1]. October 2024 saw the release of version 1.8, which focused on enhancing the scalability and performance of the Matrix network with optimizations for large-scale deployments and better database management. In December 2024, version 1.9 introduced advanced moderation tools to manage public and private rooms better and improve user identity management.

Finally, in February 2024, version 1.10 focused on security enhancements, including updates to encryption algorithms and additional protections against spam and abuse. In April 2024, version 1.11 was released, integrating new features to improve user experience, including refined user UI and more efficient communication protocols, further solidifying Matrix's position as a leading decentralized communication platform [1]. Having explored the historical development of Matrix, we now move on to examine the architecture and functionality that make it a robust decentralized platform.

Architecture and functioning of the matrix protocol

The Matrix protocol facilitates decentralized, secure, and interoperable communication across various devices and platforms. Its architecture relies on several core components and mechanisms that ensure efficient network data synchronization, user discovery, and message integrity.

General structure of the Matrix protocol

The general structure of the Matrix protocol is based on a federated architecture, where multiple independent servers, known as **homeservers**, communicate with each other to synchronize data and messages across the network. Each homeserver is an independent entity that can be run by anyone—from individuals to large organizations—allowing for a truly decentralized communication infrastructure. This architecture contrasts with traditional centralized models, where a single server or a cluster of servers controlled by a central authority handles all data and message routing.

In the Matrix ecosystem, each user is associated with a specific homeserver, their primary point of presence in the network. This homeserver stores the user's messages, profile data, contact lists, and other relevant information. When users register on the Matrix network, they choose a homeserver to register their accounts. All the user's data, including messages, media, and room memberships, are stored on this homeserver. Importantly, users on different homeservers can still communicate seamlessly because homeservers are designed to federate with one another, synchronizing data to ensure a consistent global state across the Matrix network [1]. Homeservers communicate using the **Server-Server API** (Federation API), which defines the protocols and methods for exchanging events, messages, and states between servers. This dual-API structure ensures a clear separation between client communication and server-to-server synchronization, contributing to the scalability and resilience of the network [1].

End-to-end message lifecycle and server synchronization

When a user sends a message in Matrix, the process begins on the client device. The message is first encrypted using the Olm protocol (for one-to-one communication) or Megolm (for group communication). In both cases, encryption occurs locally, and only the intended recipients hold the necessary keys to decrypt the message [1].

After encryption, the message is signed and sent to the sender's homeserver. The homeserver serializes it as a JSON event and assigns it a unique identifier. This event is stored and then propagated to other homeservers participating in the same room using the Server-Server API, which handles federation across the network [1].

Each homeserver that receives the event first verifies its digital signature, generated by the origin server using Ed25519 keys. If the signature is valid, the event is integrated into the server's local copy of the room's event history. Matrix structures this history as a Directed Acyclic Graph (DAG), where each event references one or more previous events. This model allows for asynchronous message propagation and enables each server to build a consistent causal structure of the room's evolution [3, 11, 12].

Conflicts may occur when events arrive out of order or in parallel, particularly under network delays or partitioning. In these cases, Matrix applies a deterministic state resolution algorithm. The algorithm uses attributes such as event depth, timestamps, and signature validity to determine the authoritative state of the room. This process allows all non-malicious homeservers to eventually converge on the same room state, without requiring centralized consensus [3, 11].

This mechanism ensures that Matrix supports decentralized communication with strong consistency properties. It guarantees that all participants, regardless of which server they connect to, will see a coherent and verifiable conversation history [13, 14].

Users and the client-server API

Figure 1 illustrates the basic configuration of a federated Matrix network, in which each client is connected to a dedicated server. This setup allows clients to communicate within their respective servers while also enabling cross-server interactions through federation. Each server

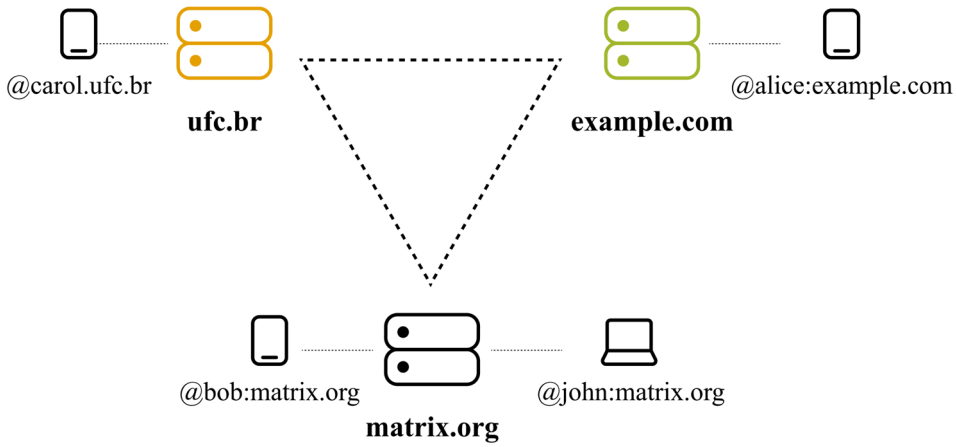


Fig. 1 Clients connected to homeservers. The servers are federated together

Table 2 Essential functions of the client-server API in the Matrix protocol

Function	Description
Continued on the next page	
User Authentication	Verifies user credentials to allow access to the network.
Message Sending and Receiving	Sends messages to and receives messages from the homeserver.
Room Management	Manages room creation, membership, and permissions.
Profile and Presence Management	Updates user profiles and displays user status.
E2EE	Encrypts messages for secure communication.
Push Notifications	Alerts users of new messages or events.
Media Uploading	Allows media file uploading and sharing.
Content Filtering	Controls incoming content visibility.

acts as an independent node, managing its own user data and handling message routing, yet remaining interconnected with other servers in the network to facilitate seamless communication across the federation.

The Matrix protocol places users at its center, interacting with the network primarily through clients—applications such as Element [6], FluffyChat [15], or Hydrogen [16]. These clients provide a user-friendly interface that allows users to send and receive messages, join rooms, manage their profiles, and communicate with other users on the Matrix network. The communication between users and the Matrix network is facilitated by the Client-Server API, which is a critical component that defines the methods and protocols used to connect client applications to homeservers.

The Client-Server API enables various essential functions that allow users to interact seamlessly with the Matrix network. Table 2 outlines the essential functions of the Client-Server API in the Matrix protocol. It highlights how client applications interact with homeservers to facilitate secure and efficient communication.

The Client-Server API manages the authentication process when a user logs in or registers on the Matrix network. It verifies the user's credentials against the homeserver database, ensuring proper authentication and authorization for accessing the network. Additionally, users can send messages to individuals or groups through their client applications. The Client-Server API transmits these messages to the user's homeserver, which then propagates them to other homeservers through federation. Incoming messages are similarly retrieved from the homeserver and displayed in real-time on the user's client. Users can manage their participation in rooms by creating, joining, or leaving rooms, with the API offering tools for managing room memberships, permissions, and administrative tasks such as inviting or banning users. The Client-Server API lets users update profile information, such as display names, avatars, and status messages. It manages presence information, indicating whether a user is online, offline, or idle, facilitating real-time communication.

The Client-Server API supports E2EE to ensure secure communications. It encrypts messages on the sender's device, allowing only the intended recipient to decrypt them, thereby guaranteeing privacy and security in all communications. Playing a key role in maintaining the integrity and efficiency of the Matrix network, the API standardizes how clients interact with homeservers and ensures compatibility across various client applications and devices. This compatibility enables users to select the client that best fits their needs without losing access to the network's full capabilities. The API further enhances

#friends: matrix.org

- {sender: "@bob:matrix.org", message: "hi"}
- {sender: "@alice:example.com", message: "morning"}
- {sender: "@carol:ufc.br", message: "hello"}

Fig. 2 The room #friends:matrix.org with simplified events

the user experience by supporting push notifications, media uploading, and content filtering features. These capabilities ensure that the Matrix ecosystem remains adaptable and can serve diverse user requirements, ranging from casual messaging to professional collaboration.

Servers and federation API

The synchronization process relies on the Server-Server API, also known as the Federation API. Servers, or homeservers, are the backbone of the Matrix protocol's decentralized architecture. Each homeserver is an independent server that stores and manages user data, including messages, media, user profiles, and room information. Homeservers are responsible for maintaining the state of the rooms where their users participate and ensuring that this state is synchronized with other homeservers across the network.

From a technical standpoint, a room consists of a sequence of JSON objects. Figure 2 illustrates a room with simplified events. The clients are responsible for correctly reading, parsing, and displaying these events.

Table 3 presents the essential events of the Server-Server API, including descriptions of each event's role in ensuring communication and synchronization between homeservers within the Matrix protocol. The events covered include Event Sending, Room Joining and Leaving, State Synchronization, and Event Authentication and Verification, highlighting their functions in maintaining consistency, integrity, and resilience across the decentralized network.

The Server-Server API defines the protocols and methods used for communication between homeservers, enabling them to exchange events, messages, and room state information. This ensures that all participating servers maintain a consistent view of the room state. When a user sends a message, it is stored on the sender's homeserver and then propagated to other homeservers via the Server-Server API. The process includes several key functions: Event Sending, Room Joining and Leaving, State Synchronization, and Event Authentication and Verification. In Event Sending, homeservers share events, such as messages or state updates, with other homeservers that have users participating in the same room, ensuring that all involved servers have the latest events. For Room Joining and Leaving, when a user joins or exits a room, their homeserver informs the other homeservers to ensure the room state is updated network-wide.

In the State Synchronization process, homeservers periodically synchronize room states to maintain consistency, comparing states, resolving conflicts, and agreeing on a common event history through a DAG model. Additionally, Event Authentication and Verification ensures the integrity and authenticity of events received from other homeservers. Each homeserver verifies events using cryptographic signatures, confirming that the data has not been altered during transmission and comes from legitimate sources. This combination of processes helps maintain a secure, synchronized, and consistent network across all homeservers.

The Server-Server API is essential for enabling federation, the core mechanism that allows different homeservers to communicate and collaborate in a decentralized manner. By facilitating seamless synchronization and communication between homeservers, the Server-Server API ensures that the Matrix network remains resilient, scalable, and consistent, even during network disruptions or server outages [1].

Figure 3 demonstrates three homeservers maintaining synchronization of their local copies of a room. In this configuration, each homeserver keeps its own version of the room's state, ensuring that all updates are propagated across the servers. When an event occurs in the room, it is shared among the homeservers, which then update their local states accordingly to remain in sync.

Homeservers operated by different individuals, organizations, or communities can coexist and communicate within the Matrix ecosystem. This setup maintains the integrity and continuity of user communications across a distributed network.

Identity servers and user discovery

Identity servers in the Matrix ecosystem play a crucial role in user discovery and connectivity by linking Matrix IDs (e.g., `@user:matrix.org`) to third-party identifiers, such as email addresses or phone numbers. This

Table 3 Essential events of the server-server API

Event	Description
Event Sending	Homeservers share events like messages and updates to keep room states consistent across servers.
Room Joining and Leaving	Homeservers update other servers when users join or leave rooms to maintain accurate room states.
State Synchronization	Homeservers periodically compare and synchronize room states using a Directed Acyclic Graph (DAG) model.
Event Authentication and Verification	Homeservers verify events from others using cryptographic signatures to ensure authenticity and integrity.

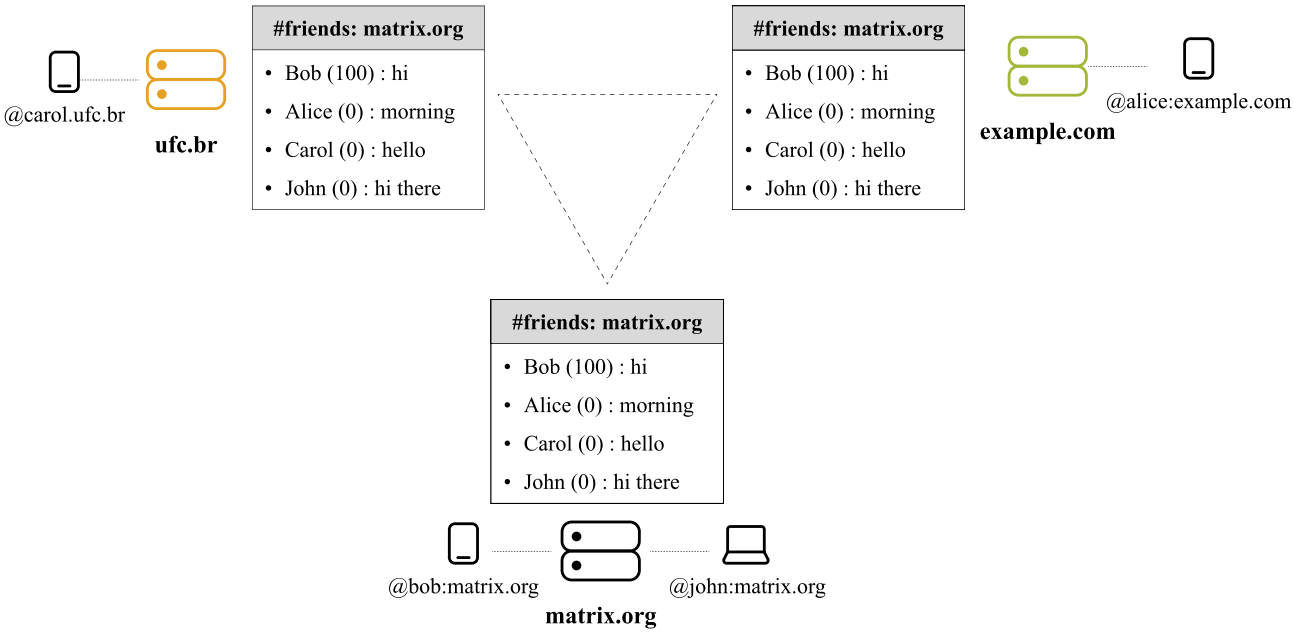


Fig. 3 Multiple homeservers maintain synchronization of their local room copies

Table 4 Main functions of identity servers and user discovery

Function	Description
User Discovery	Helps users find each other on the Matrix network by mapping third-party identifiers (e.g., email addresses, phone numbers) to Matrix IDs.
Contact Lookup	Allows users to check if their contacts are on Matrix by querying linked identifiers.
Privacy and Consent	Manages user consent for linking identifiers, ensuring privacy and compliance with data protection regulations.
Decentralization Support	Facilitates user discovery without relying on centralized services, supporting Matrix's decentralized nature.

functionality allows users to easily find each other on the Matrix network by searching with familiar identifiers, which the identity server then maps to the corresponding Matrix ID. This mapping service is especially valuable for onboarding new users who might not be familiar with Matrix ID formats or when integrating existing contact lists from other platforms [1]. The use of identity servers is particularly useful for simplifying user interactions and enhancing connectivity; however, it also raises important privacy considerations. To address these, Matrix allows multiple identity servers to operate independently, giving users the choice of which server to trust and ensuring they have control over their data. This decentralized approach helps safeguard user privacy and aligns with Matrix's broader goals of decentralization and resilience against censorship. These functions collectively contribute to a robust, privacy-respecting, and user-friendly environment, aligning with Matrix's commitment to open and decentralized communication. By facilitating seamless user discovery, efficient contact lookup, stringent privacy management, and support for decentralized operations, identity servers significantly enhance the functionality and appeal of the Matrix protocol. Table 4 describes these functions in detail, highlighting their importance in maintaining the integrity and usability of the Matrix network.

Event-based communication model

Matrix operates on an event-based communication model, where every action within the network is captured as an event. These events cover a wide range of actions, including sending messages, joining rooms, updating profiles, or modifying room settings. Each event is carefully timestamped and stored in chronological order, ensuring that all participants in a room observe the same sequence of messages and actions, regardless of the homeserver they are connected to. This structure guarantees a consistent and synchronized experience across the network, allowing users to follow the conversation's flow seamlessly, no matter their location or server connection.

In Matrix, events are represented as immutable objects within a DAG. The DAG structure enables each event to reference one or more preceding events, creating a chain

that establishes the order of occurrences. This model allows homeservers to maintain a coherent view of the room's state without requiring a central authority to manage the sequence of actions. As a result, the Matrix network ensures data integrity and coherence across its distributed architecture, supporting a decentralized and resilient system for communication. The event-based DAG model enhances reliability and transparency, enabling efficient collaboration and secure interactions throughout the network.

Decentralization

Decentralization is a core principle of the Matrix protocol. Unlike traditional centralized messaging platforms, where all data is controlled by a single entity (e.g., a company that owns the messaging service), Matrix allows anyone to run their own homeserver. This means that users have control over their data, and the network is not dependent on any single point of failure [1]. Matrix's decentralized architecture enhances resilience against outages and censorship. If one homeserver goes offline, the data stored on other homeservers remains intact, and communication can continue. This makes Matrix particularly appealing in environments where censorship is a concern, as it is much harder to suppress information on a decentralized network [1].

Federation plays a crucial role in decentralization by allowing homeservers to communicate and synchronize data with each other. Each homeserver in the network is autonomous, yet they cooperate to maintain a consistent and coherent view of the communication across the network. This cooperation ensures that users on different homeservers can interact as if they were on the same server [1]. Furthermore, Matrix supports peer-to-peer (P2P) communication, where users can communicate directly without the need for a homeserver. This is particularly useful in scenarios where infrastructure is limited or where users want to avoid relying on third-party servers altogether [1].

Security

Security is one of the main pillars of the Matrix protocol, with E2EE being a critical feature. E2EE ensures that messages are encrypted locally before transmission. Only the recipient, holding the correct decryption key, can access the original content. Even if intercepted during transfer, the content of the communication remains secure and unreadable by unauthorized parties [1]. Table 5 summarizes the cryptographic functions employed by the Matrix protocol, detailing its core mechanisms. Each key type – Ed25519, Curve25519, Olm, and Megolm – plays a role in securing and maintaining the integrity of communications within the Matrix network.

Matrix uses the Olm and Megolm cryptographic ratchets to implement E2EE. Olm is used for one-to-one communication, while Megolm is designed for group communication. These ratchets ensure forward secrecy, meaning that even if one encryption key is compromised, past and future communications remain secure [1]. The Olm and Megolm protocols rely on cryptographic primitives like Ed25519 and Curve25519 to achieve their security properties. Ed25519 is used for digital signatures, providing a secure method to authenticate the sender of a message. Curve25519, on the other hand, is used for key exchange, allowing two parties to securely establish a shared secret over an insecure channel [17].

In addition to E2EE, Matrix employs other security measures, such as device verification and cross-signing. Device verification allows users to confirm the authenticity of each other's devices by comparing unique device fingerprints. This process ensures that users communicate with the correct devices, preventing man-in-the-middle (MITM) attacks. Cross-signing enables users to verify a new device once and have that verification apply to all their other devices automatically, streamlining the security process across multiple devices [1]. Matrix also prioritizes transparency and auditability. The protocol's decentralized nature ensures that no single entity can alter or delete data without consensus, and all actions

Table 5 Functions of keys in the Matrix protocol

Key Type	Function
Ed25519	Used for digital signatures.
	Ensures the authenticity of the sender.
	Provides message integrity by verifying that the content has not been tampered with.
Curve25519	Utilized for key exchange.
	Allows two devices to establish a shared secret securely.
	Fundamental for setting up encrypted communication channels between devices.
Olm	Implements a ratcheting protocol for E2EE in one-to-one communications.
	Combines Curve25519 for key exchanges and Ed25519 for signatures.
	Provides forward secrecy, post-compromised security, and deniability.
Megolm	An adaptation of the Olm protocol designed for group communications.
	Uses a single ratchet shared among all group members.
	Allows efficient encryption while maintaining forward secrecy and message integrity.

are recorded as events, providing a clear audit trail. This auditability is crucial for maintaining trust in a decentralized system, as it allows users to verify that their data has not been tampered with [1].

Security boundaries in federated environments

Matrix allows users to operate their own homeservers, which increases autonomy over infrastructure and key management. However, this does not imply full data ownership. Due to federation, messages are often replicated to other homeservers participating in the same room. This replication reduces the level of control an individual user has over their data, particularly in public or multi-party conversations [18].

Forward secrecy in Matrix ensures that past messages remain confidential even if current or future encryption keys are compromised. However, this mechanism does not protect future communication. It prevents retrospective decryption but not prediction or interception of messages yet to be sent [1].

Matrix does not offer strong metadata protection by default. Elements such as message timestamps, room membership, and homeserver domains can be inferred through traffic analysis [19]. While message contents remain encrypted, metadata exposure remains a challenge for privacy-sensitive use cases. On the client side, protection mechanisms vary across implementations. Typically, encrypted messages are stored locally, and key handling is done using secure enclaves or encrypted key stores when available.

Security mechanisms and future considerations

Table 6 summarizes the main cryptographic mechanisms used in the Matrix protocol. Each algorithm supports a specific security property, such as confidentiality, integrity, or authentication. This classification provides a clearer view of how security goals are achieved across different layers of the protocol.

Many of these mechanisms were introduced or improved in specific protocol versions. End-to-end encryption (E2EE) became a standard feature in version 1.0. Improvements in device verification and key management appeared in later versions, such as 1.1 and 1.3. The most recent releases (1.10 and 1.11) introduced updates to encryption handling and protections against abuse.

Matrix currently uses Ed25519 and Curve25519 for signatures and key exchanges. These elliptic curves are expected to become deprecated after 2035, due to their vulnerability to quantum computing attacks [20]. Although the Matrix specification does not yet define a post-quantum transition, discussions in the open-source community have explored alternatives such as CRYSTALS-Dilithium and Kyber, which are among the algorithms selected by NIST for standardization [21].

Future versions of the protocol may incorporate hybrid or post-quantum schemes to mitigate long-term risks. However, challenges remain in balancing security, performance, and compatibility across a wide range of client implementations.

Interoperability

Interoperability is a fundamental goal of the Matrix protocol, enabling it to bridge communication silos and allow users on different platforms to interact seamlessly. Matrix was designed to connect disparate communication systems, making it possible for users on Matrix to communicate with users on other platforms like Slack, Discord, IRC, or even traditional SMS [1].

Matrix achieves interoperability through the use of bridges. Bridges are connectors (Fig. 4) that translate data and protocols between Matrix and other communication systems. For example, a Matrix-Slack bridge would allow users in a Matrix room to communicate with users in your network as if they were in the same chat room. This bridging capability is one of Matrix's most powerful features, as it creates a unified communication experience across different platforms [1].

In addition to bridging, Matrix's open standard and API-driven architecture make it easy for developers to build custom integrations and applications on top of the protocol. This extensibility has led to developing a wide range of clients, bots, and services that enhance the Matrix ecosystem. Its support for other standards and protocols also reflects Matrix's commitment to openness and interoperability. For example, Matrix can integrate with WebRTC for real-time voice and video communication, and it supports the integration of decentralized identity solutions, making it a versatile platform for modern communication needs [1].

Table 6 Cryptographic mechanisms in Matrix and their purposes

Purpose	Mechanism	Details
Confidentiality	Olm / Megolm	End-to-end encryption of messages
Key Exchange	Curve25519	Derives shared secrets between devices
Integrity / Authentication	Ed25519 (digital signatures)	Event signing and verification
Forward Secrecy	Olm (1:1)	Per-message ratcheting
Group Messaging	Megolm	Shared symmetric ratchet for groups

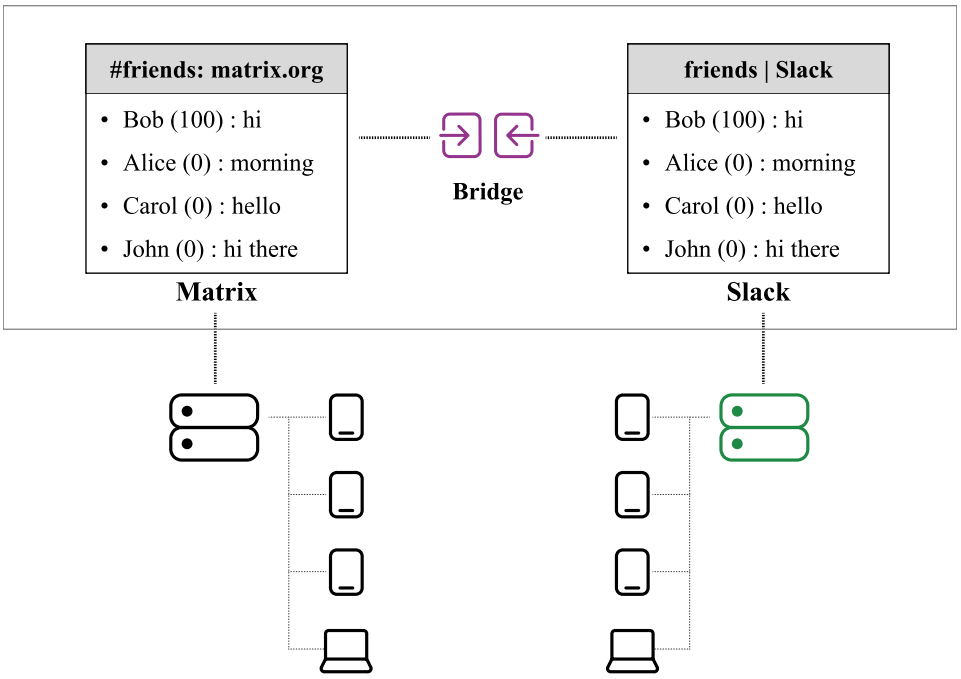


Fig. 4 A room bridged between matrix.org and slack.com

Table 7 Typical use cases of the Matrix protocol

Use Case	Description
Instant Messaging	Secure, encrypted text messaging between users.
Private Group Chat	Secure spaces for group messaging and file sharing.
Public Rooms	Open forums for discussions, support, and events.
VoIP Signaling	Secure voice and video calls using Matrix's messaging mechanism.
Internet of Things (IoT) Communication	Secure data exchange between IoT devices in a decentralized manner.
Bridging Existing Communication Silos	Connecting different platforms like Slack, Discord, Telegram, and IRC for seamless communication.

Use and applications of the matrix protocol

The Matrix protocol was designed to provide a flexible and secure platform for decentralized communication. Since its launch, it has been adopted in various use cases, from instant messaging and group chats to online communities and integration with other communication platforms. In this section, we will discuss some of the typical use cases of Matrix, examples of platforms and services that utilize the protocol, and the benefits and challenges of adopting Matrix.

Typical use cases of Matrix

Matrix's use cases are broad and diverse, reflecting its flexibility and ability to adapt to different communication needs. Table 7 outlines the typical use cases of the Matrix protocol, showcasing its versatility in supporting various communication needs. It includes applications ranging from instant messaging and private group chats to public rooms for large-scale interactions. Additionally, the protocol facilitates secure voice and video calls. It enables communication between IoT devices and bridges different communication platforms. These capabilities highlight Matrix's adaptability and strength in fostering decentralized, safe, and interconnected communication environments.

Matrix serves as an instant messaging platform, allowing users to exchange text messages quickly and securely through E2EE, ensuring that only participants can access the content of conversations. Beyond individual messaging, Matrix enables the creation of private group chats where users can share messages and files in a protected environment, making it ideal for work teams, families, or any group needing a secure space for communication. In addition to private chats, Matrix supports public rooms where any user can join and participate in discussions. These rooms are commonly used for forums, technical support, or live events, providing an organized space for large groups to interact. Another notable feature is Matrix's use for VoIP signaling, which enables secure and decentralized voice and video calls. The same event mechanism used for messaging also facilitates calls, ensuring encryption and privacy, making Matrix a strong option for real-time multimedia communication.

Moreover, Matrix's capabilities extend to communication between IoT devices, allowing them to exchange data securely without relying on a central server. This decentralized approach is crucial for scenarios where privacy and data security are paramount. Additionally, Matrix excels at bridging communication silos by connecting users across different platforms like Slack, Discord, Telegram, and IRC networks. By acting as a bridge, Matrix allows seamless communication across previously isolated systems, making it a versatile solution for both personal and professional use. The flexibility in connecting various platforms and supporting secure communication makes Matrix a comprehensive tool for diverse communication needs.

Communities in Matrix

The concept of communities in Matrix stands out as one of the protocol's most powerful and flexible features. Communities allow users to create and share spaces that can group multiple rooms, streamlining the organization and moderation of large-scale discussions. In Matrix, spaces serve as collections of rooms that can be arranged into hierarchies, enabling more efficient management of interactions. A space can represent various entities, such as a community, project team, or interest group. Space creators are given the tools to add rooms, set permissions, and invite members, establishing a collaborative and structured environment where communication can flourish.

Space administrators play a key role in maintaining order within these communities. They have the ability to add new rooms, assign moderators, and set rules for participation. Basic moderation tools allow administrators to ban users, set discussion topics, and control access to rooms, ensuring the community remains organized and secure. This level of control enables communities to scale effectively while maintaining a safe and well-structured environment for participants to interact. Through these capabilities, Matrix provides an adaptable solution for creating, managing, and moderating large-scale conversations across various interest groups and projects.

Matrix ecosystem

The Matrix ecosystem is vast and diverse, with many clients, servers, and bridges extending its capabilities. Figure 5 illustrates the Matrix ecosystem, highlighting its key components and their interactions. Label 1 represents the users or clients who interact with the homeservers. Label 2 identifies the homeservers, which are responsible for managing communication and data storage for clients, while also federating with other homeservers to maintain a decentralized and interconnected network. Label 3 points to identity servers, which handle user identification and authentication within the Matrix network. Label 4 indicates service applications, such as bridges, which facilitate connections between Matrix and other communication platforms. Finally, Label 5 represents communication service providers that can integrate with and access the Matrix network, broadening

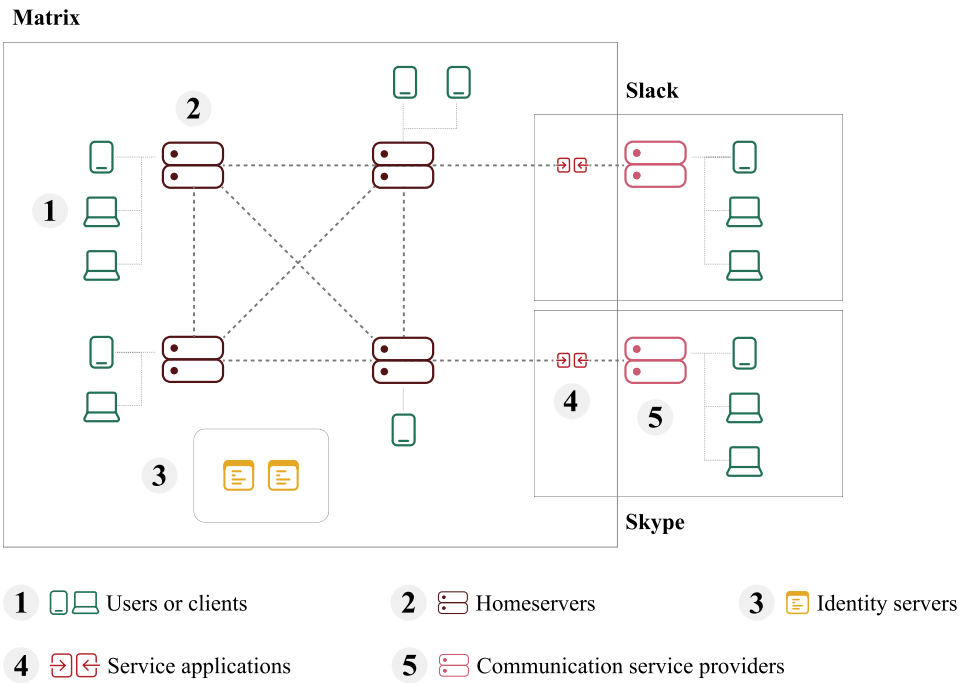


Fig. 5 The Matrix ecosystem with all its components, including clients and servers

Table 8 Key components of the Matrix ecosystem

Component	Description
Clients	Element, Element X, FluffyChat, and Hydrogen.
Servers	Synapse, Dendrite, and Conduit.
Bridges	Slack, Discord, and IRC.

Table 9 Related work comparison

Ref.no.	Year	Type of Research	Focus	Considered Applications	Objective
[26]	2021	Survey	Security	Signal, WhatsApp, Wire, Viber, Riot, Telegram	Compare applications
	2023	Survey	Usability	Mattermost, Matrix, Element, Cinny, Gitter, Zulip, Twake, Rocket.Chat, Corteza, Revolt	Compare applications
[27]	2023	Survey	Interoperability	Signal, WhatsApp, Matrix	How to integrate applications
[28]	2023	Survey	Security	Element, WhatsApp	Compare applications
[4]	2023	Survey	Security	Element, Facebook Messenger, FaceTime, Signal, Telegram, WhatsApp, and others	Compare applications
This paper	2024	Sistematic Mapping Study	Relationship of the paper with Matrix, area, main contribution, explored Matrix feature	Totally focused on Matrix	Analyze the Matrix

the scope of connectivity and interoperability within the ecosystem. This comprehensive setup underlines Matrix's decentralized and federated nature, allowing seamless and secure communication across different services and platforms.

Table 8 presents the key components of the Matrix ecosystem, highlighting the variety of clients, servers, and bridges that contribute to its versatility and interoperability. The Matrix protocol supports diverse clients, from the widely recognized Element to other alternatives like FluffyChat [15], Hydrogen [6], and SchildiChat [22], each offering unique user experiences. Matrix servers such as Synapse [23], Dendrite [24], and Conduit [25] also form the network's backbone, catering to different performance and feature requirements. Moreover, Matrix's ability to bridge connections with platforms like Slack, Discord, Telegram, and IRC underscores its commitment to seamless cross-platform communication.

Related work

In the rapidly evolving field of communication and messaging systems, a substantial body of research has emerged, addressing various aspects such as security, usability, and interoperability. This section reviews key studies that have shaped our understanding of these topics, as summarized in Table 9. The table provides a comparative overview of significant papers that have performed some type of literature review in this topic. The comparison details their year of publication, type of research, focus, considered applications and objective. By examining these contributions, we highlight the progression of research and identify the specific niche our study addresses. Our research stands out by offering a detailed mapping study of the Matrix protocol – a topic that has not been extensively explored in prior work. Following, we discuss each table's column.

Year. Research in communication and messaging systems has seen considerable growth and diversification in recent years. Foundational studies began as early as 2021, laying the groundwork for deeper exploration into various aspects of these technologies. The year 2023 was particularly noteworthy, with a surge of significant research contributions, such as those by Andriessen et al. and Blessing et al. [27], reflecting heightened interest and activity in the field. Our study, developed in 2024, builds upon this momentum by delivering a focused analysis of the Matrix protocol. The Matrix protocol is increasingly recognized for its potential to enhance interoperability and security in decentralized communication systems. Our paper highlights the ongoing evolution and refinement of this protocol, offering fresh insights and contributing to its growing importance within the broader landscape of messaging systems.

Type of Research. The body of research in communication and messaging systems includes various methodological approaches, each serving distinct purposes. Surveys, as seen in papers like Johansen et al. [26] and Blessing et al. [27], offer comprehensive overviews of existing technologies and practices, often focusing on comparisons across multiple platforms. These studies are valuable for understanding general trends and benchmarks within the field. However, they typically provide limited, not systematic, high-level insights rather than detailed analyses of specific technologies. In contrast,

our study utilizes a systematic mapping study approach [29]. This methodology is crucial for thoroughly examining a particular technology's features, contributions, and its role within the broader ecosystem. By systematically mapping the Matrix protocol, our research fills a critical gap by providing a detailed, structured analysis that highlights its unique aspects and contributions, which are often overlooked in more general surveys. This approach allows us to offer deeper insights into the Matrix protocol.

Focus. Previous research on communication and messaging systems has predominantly concentrated on critical areas such as security, usability, and interoperability. For instance, studies by Johansen et al. [26] and Blessing et al. [28] focus heavily on security, highlighting the ongoing concern with safeguarding communication data. Usability is the primary focus of Andriessen et al. , which addresses how user-friendly different systems are. Additionally, Blessing et al. [27] concentrate on interoperability, examining how well various systems integrate with one another. In contrast, our research concentrates on the Matrix protocol. This targeted analysis offers new insights into the Matrix protocol's role and performance that are not as thoroughly examined in the existing literature.

Considered Applications. The applications examined in the literature vary widely, including Signal, WhatsApp, Matrix, and others. Research such as Andriessen et al. and Alatawi et al. [4] covers a broad range of platforms, while Blessing et al. [27] focuses on fewer applications. Our study stands out by concentrating exclusively on the Matrix protocol, offering a comprehensive analysis of its capabilities and contributions. This targeted examination provides a detailed understanding of Matrix that contrasts with the broader application scopes of other studies.

Objective. The objectives outlined in the table reflect a range of approaches to understanding and evaluating communication and messaging systems. Most prior research, including the surveys by Johansen et al. [26], Andriessen et al. , and Blessing et al. [27], aim to compare various applications across different criteria, such as security, usability, and interoperability. These studies generally focus on providing a broad overview and benchmarking of multiple systems to understand their relative strengths and weaknesses. In contrast, our research, takes a distinctive approach by conducting a systematic mapping study specifically focused on the Matrix protocol. The objective of our study is to offer an in-depth analysis of the Matrix protocol, exploring its key features, contributions, and the relationships it forms within the messaging landscape. By concentrating on this single protocol, our paper seeks to provide a detailed and comprehensive understanding of Matrix.

How has the Matrix protocol been used in academic research?

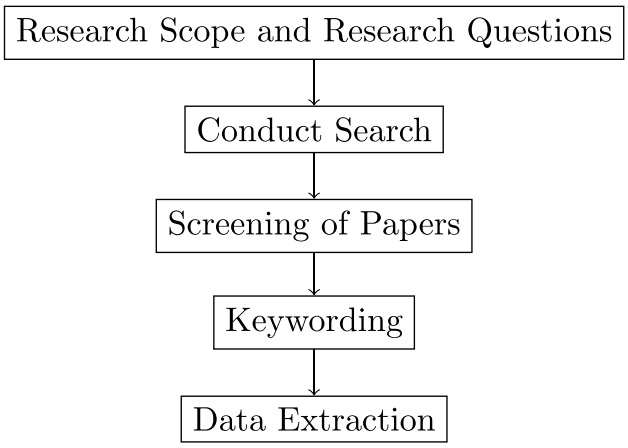


Fig. 6 Steps of the mapping study conducted in this research

Methodology

This section outlines the methodology employed in this work. We utilized a systematic mapping study to explore the topic of interest. According to Kitchenham et al. [29], a systematic mapping study involves "a broad review of primary studies in a specific topic area that aims to identify what evidence is available on the topic". Systematic mapping studies provide several benefits. They offer a comprehensive overview of a research area, highlight gaps in the existing research, and support evidence-based decision-making. By systematically reviewing the literature, researchers can assess the current state of knowledge, pinpoint areas needing further investigation, and guide future research efforts. These studies help to synthesize large amounts of information, reveal trends and patterns, and identify inconsistencies across different studies. They are valuable resources for policy-makers, practitioners, and researchers. Moreover, systematic mapping studies can reduce research duplication, improve resource allocation, and foster collaboration among researchers in related fields. We adopted the systematic mapping study process as outlined by Peterson et al. [30], which is illustrated in Fig. 6 and detailed in the subsequent subsections.

Research Scope and Research Questions

Understanding the cutting-edge methods for evaluating the Matrix protocol's performance is essential in today's digital landscape, where the adoption of decentralized communication systems is rapidly growing. As organizations increasingly implement the Matrix protocol to facilitate secure and scalable communication, evaluating its performance becomes critical. The swift advancements in decentralized technologies introduce complexities that necessitate novel evaluation methods and tools. By

Table 10 List of research questions

Question IDs	Description
RQ01	What is the relationship between the paper and the Matrix protocol?
RQ02	What is the main research area of the paper?
RQ03	What is the main contribution of the paper?
RQ04	What is the explored Matrix protocol feature in the paper?

examining the current state-of-the-art areas, contributions, and Matrix features, we can gain insights into its performance and identify areas needing improvement. This exploration will help uncover innovative evaluation strategies and highlight gaps in the existing literature, providing a solid foundation for future research aimed at optimizing the Matrix protocol's performance and functionality. Based on these considerations, this paper seeks to answer the following question:

We can delve into more specific inquiries stemming from the general question. Our focus is on addressing four key questions outlined in the Table 10.

RQ01. Understanding the *relationship between research papers and the Matrix protocol* is essential for evaluating how the protocol integrates into the broader academic and practical discussions on decentralized communication. The Matrix protocol, known for its robust framework supporting decentralized and secure messaging, has been increasingly influential across various research domains, including cybersecurity, distributed systems, and real-time communication. By examining this relationship, we can understand how scholars and practitioners are integrating the Matrix protocol into their research. This may involve focusing on the protocol itself or using it as a component in broader studies of decentralized technologies. This analysis not only helps to trace the protocol's impact on the field but also reveals how it influences and is shaped by ongoing research efforts, offering a clearer understanding of its role in advancing communication technologies.

RQ02. Identifying the primary *research area of papers* that explore the Matrix protocol is crucial for contextualizing its application and relevance. The Matrix protocol intersects with several research domains, such as cybersecurity, real-time communication, and network architecture. By determining the main research area, we can better understand how the Matrix protocol is applied within these fields and assess its contributions to solving specific problems or advancing knowledge in these domains. This clarity helps in recognizing trends and emerging challenges related to the protocol and provides direction for future research efforts aimed at leveraging or enhancing its capabilities.

RQ03. Evaluating the *main contribution of the papers* that investigates the Matrix protocol highlight the advancements and innovations it brings to the field. Contributions can range from introducing new evaluation methodologies for the protocol, proposing enhancements to its existing features, or providing novel insights into its practical applications and performance. Understanding these contributions allows us to appreciate how the paper advances the development and deployment of the Matrix protocol. It also helps in identifying specific areas where the protocol is being improved or expanded, which is crucial for addressing gaps in current research and guiding future investigations. This knowledge is vital for recognizing the paper's impact on both theoretical and practical aspects of decentralized communication systems.

RQ04. Investigating the *specific feature of the Matrix protocol* that a paper explores provides a detailed understanding of the focus and scope of the research. The Matrix protocol encompasses a variety of features such as E2EE for secure messaging, interoperability with other communication systems, and decentralized data storage. By identifying which feature is the primary focus of the paper, we can assess how it addresses specific challenges or opportunities related to that feature. This insight helps in understanding the current priorities in research on the Matrix protocol and highlights areas where further development or investigation may be needed. It also provides a deeper appreciation of how different aspects of the protocol are being examined and optimized, contributing to the overall evolution of decentralized communication technologies.

Conduct search (For Primary Studies)

We conducted a search across the electronic sources listed in Table 11, focusing on publications that cited terms related to the Matrix protocol, such as "Matrix protocol," "matrix.org," and other related variations. We chose to include "matrix.org" in our search because it is the primary repository for the Matrix protocol's documentation. Our search strategy involved examining the full text of these publications, which enabled us to identify papers specifically addressing the Matrix protocol, including those discussing its key features such as security, decentralization, and chat mechanisms.

Screening of papers (For inclusion and exclusion)

Inclusion and exclusion criteria are indispensable elements of mapping studies, serving as a navigational compass that guides researchers through the vast sea of literature. These criteria ensure precision in research

Table 11 Search strings

Search Engines	Quantity	Search String
IEEE Xplore	16	"matrix.org" OR "matrix protocol" OR "protocol matrix"
ACM Digital Library	20	[All: "matrix protocol"] OR [All: "protocol matrix"] OR [All: "matrix.org"]
Google Scholar	12	Search for "matrix.org", "matrix protocol", "protocol matrix" then click on Sort by date - Abstracts
ScienceDirect	82	Title, abstract, keywords, References: "matrix.org" OR "matrix protocol" OR "protocol matrix"
SpringerLink	26	"matrix.org" OR "matrix protocol" OR "protocol matrix"
TOTAL	**156**	

Table 12 Inclusion and exclusion criteria

Inclusion	Exclusion
• The abstract or title explicitly mentions the Matrix protocol.	• The paper does not conduct experiments, analyses, comparisons, or correlations using the protocol Matrix.
• Perform a practical experiment with the Matrix protocol.	• Not written in English.
• Performs formal analysis on the Matrix protocol.	
• Only Mentions through comparison or correlation.	

Table 13 Primary studies detailed as references

Ref.no.	Title
[12]	"A Glimpse of the Matrix: Scalability Issues of a New Message-Oriented Data Synchronization Middleware", Jacob, F. et al., 2019, Middleware'19.
[3]	"Matrix Decomposition: Analysis of an Access Control Approach on Transaction-Based DAGs without Finality", Jacob, F. et al., 2020, SACMAT'20.
[11]	"Analysis of the Matrix Event Graph Replicated Data Type", Jacob, F. et al., 2021, IEEE Access.
[31]	"Help, My Signal Has Bad Device!", Wichelmann, J. et al., 2021, DIMVA 2021.
[9]	"Key Agreement for Decentralized Secure Group Messaging", Weidner, M. et al., 2021, CCS 2021.
[32]	"Pak e-Mail: Authentication and Key Management in Decentralized Secure Email", Vazquez Sandoval, I. et al., 2021.
[2]	"Forensic Analysis of Matrix Protocol and Riot.im Application", Schipper, G.C. et al., 2021, Forensic Science Int.
[33]	"Extracting Entities from Production Chats", Müller, M., 2021, Procedia CIRP.
[34]	"Safe and Private Forward-Trading Platform for Microgrids", Eisele, S. et al., 2021.
[19]	"DRIFT: E2EE Spatial Feature Sharing & Messaging", Brunila, M. et al., 2022, SIGSPATIAL.
[35]	"teacher.js: Low-bandwidth Tool for Remote Teaching", Brodbeck, F. et al., 2022, IEEE ECEI.
[36]	"Towards Portable Identities in the Matrix Protocol", Ihle, C. et al., 2022, ICDCSW.
[37]	"Impact of WhatsApp's 2021 Privacy Policy", Griggio, C.F. et al., 2022, CHI 2022.
[38]	"Threat Models of E2EE Messaging Apps", Chowdhury, P.D. et al., 2023, arXiv.
[39]	"Fork-Resilient Group Key Agreement", Alwen, J. et al., 2023, CRYPTO 2023.
[40]	"Where on Earth is the Spatial Name System?", Gibb, R. et al., 2023, HotNets'23.
[14]	"Demystifying Decentralized Matrix Communication", Li, H. et al., 2023, ICPADS.
[13]	"Cryptographic Vulnerabilities in Matrix", Albrecht, M.R. et al., 2023, IEEE S&P.
[41]	"Comparse: Secure Formats for Cryptographic Protocols", Wallez, T. et al., 2023, CCS 2023.
[42]	"CrossChat: Messaging Across Different Apps", Cui, M. et al., 2023, ACM SAC.
[43]	"Matrix-Mesh Communication Protocol", Rahman, M. et al., 2024, Dew Computing.

scope by delineating the boundaries of relevance, allowing scholars to focus exclusively on literature that directly addresses their research questions and objectives. They minimize bias by providing transparent guidelines for selection and optimize resource utilization by efficiently allocating time and effort to synthesizing high-quality evidence. Moreover, adherence to these criteria improves the reproducibility and transparency of mapping studies, enabling other researchers to validate findings and build upon existing knowledge. In essence, inclusion and exclusion criteria are the cornerstone of methodological rigor in mapping studies. They facilitate the synthesis of meaningful insights that contribute to the advancement of scholarly discourse and understanding in various fields. Table 12 presents the inclusion and exclusion criteria for this work. Table 13 shows the 21 selected studies.

Table 14 Unified keywords

Classification	Keywords
Relationship Between the Article and The Matrix	Uses the Matrix Protocol in Practice; Only Mentions.
Main Area of the Paper	Communication and Networks; Security and Privacy; Forensic Analysis; Natural Language Processing (NLP); Educational Technology; Human-Computer Interaction (HCI).
Main Contribution	Matrix Protocol Analysis; Security and Privacy Approach; Protocol Development; Tool Development.
Explored Matrix Feature	Group Communication; Security Issues; Matrix Architecture.

Table 15 Categorization of each paper

Ref.no.	Relationship between the paper and the Matrix	Main area of the paper	Main contribution	Explored Matrix feature
[12]	Uses the Matrix Protocol in Practice	Communication and Networks	Matrix Protocol Analysis	Group Communication
[3]	Uses the Matrix Protocol in Practice	Communication and Networks and Security and Privacy	Matrix Protocol Analysis / Security and privacy Approach	Matrix Architecture
[11]	Uses the Matrix Protocol in Practice	Communication and Networks and Security and Privacy	Matrix Protocol Analysis / Security and privacy Approach	Matrix Architecture
[31]	Only Mentions	Security and Privacy	Security and privacy Approach	Security Issues
[9]	Uses the Matrix Protocol in Practice	Security and Privacy	Protocol development	Security Issues
[32]	Only Mentions	Security and Privacy	Protocol development	Group Communication
[2]	Uses the Matrix Protocol in Practice	Forensic analysis	Matrix Protocol Analysis	Group Communication
[33]	Uses the Matrix Protocol in Practice	Natural Language Processing	Tool development	Group Communication
[34]	Only Mentions	Security and Privacy	Tool development	Security Issues
[19]	Uses the Matrix Protocol in Practice	Security and Privacy	Tool development	Group Communication and Security Issues
[35]	Uses the Matrix Protocol in Practice	Educational Technology	Tool development	Group Communication
[36]	Uses the Matrix Protocol in Practice	Security and Privacy	Security and privacy Approach	Group Communication
[37]	Only Mentions	HCI	Security and privacy Approach	Group Communication
[38]	Uses the Matrix Protocol in Practice	Security and Privacy	Security and privacy Approach	Security Issues
[39]	Only Mentions	Communication and Networks	Protocol development	Security Issues
[40]	Only Mentions	Communication and Networks	Protocol development	Group Communication
[14]	Uses the Matrix Protocol in Practice	Security and Privacy	Matrix Protocol Analysis Security / privacy Approach	Matrix Architecture
[13]	Uses the Matrix Protocol in Practice	Security and Privacy	Matrix Protocol Analysis / Security and privacy Approach	Matrix Architecture
[41]	Only Mentions	Security and Privacy	Tool development	Security Issues
[42]	Only Mentions	Communication and Networks / HCI	Tool development	Group Communication
[43]	Uses the Matrix Protocol in Practice	Communication and Networks	Protocol development	Matrix Architecture

Keywording

We have found 15 expressions, which were grouped and presented in Table 14. The expressions were unified when they appeared with similar meanings, as suggested by Peterson et al. [30]. The analysis categorized papers based on their relationship to the Matrix protocol, distinguishing between those that use the protocol in practice and those that merely mention it. We also classified the main research areas covered in the literature, which include Communication and Networks, Security and Privacy, Forensic Analysis, Natural Language Processing (NLP), Educational Technology, and Human-Computer Interaction (HCI). Furthermore, the main contributions of the papers were identified as Matrix Protocol Analysis, Security and Privacy Approaches, Protocol Development, and Tool Development. Finally, we explored specific Matrix features addressed in the research, such as Group Communication, Security Issues, and Matrix Architecture. This classification allowed us to comprehensively understand the scope and impact of the Matrix protocol within various research domains and its specific applications and contributions.

Data extraction

In this step, we associate the keywords to each paper, producing the results listed in Table 15. This table allows the reader to access the specific paper that addresses the desired characteristics.

Table 15 synthesizes the main findings of this mapping study. It classifies each primary study along four dimensions: its relationship with the Matrix protocol, the main research area, the type of contribution, and the specific

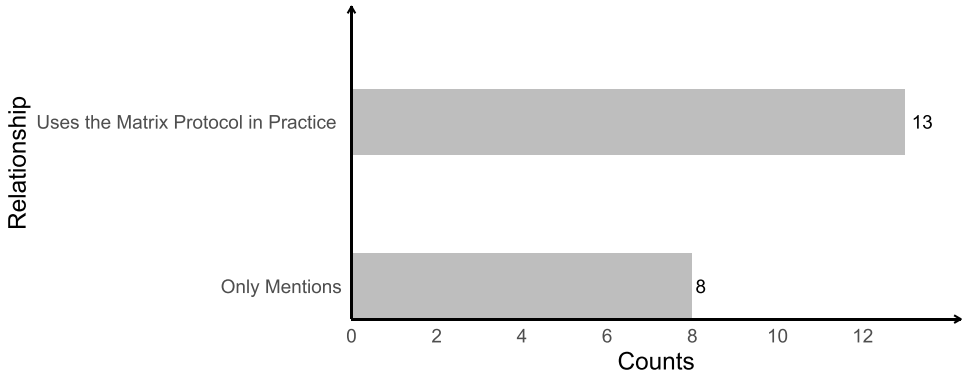


Fig. 7 Primary studies quantified by relationship

protocol features addressed. This structured classification enables a clearer understanding of how Matrix has been used and studied across different domains.

The table helps identify trends, such as the dominance of security-focused research, and reveals gaps in under-explored areas like usability and interoperability. It also supports comparisons across studies, showing overlaps and divergences in methodology and focus. Due to its integrative value, we consider Table 15 a significant contribution of this work, as it provides a reusable reference for researchers and practitioners working on decentralized communication systems.

Research answers

Now, we will analyze each of the research questions one by one. First, we will explain the meaning of each keyword used to categorize the papers. Next, the information will be presented in graphical format.

RQ01 - Relationship with Matrix

Figure 7 shows the relationship between the papers and the Matrix protocol. The papers were classified as **Uses the Matrix Protocol in Practice** and **Only Mentions**.

Thirteen papers have **used the Matrix protocol in practice**. These papers use the Matrix protocol in experiments based on code implementation or through formal analysis. We believe that, given that Matrix is a set of open APIs for federated and open communication of Instant Messaging (IM), VoIP, and IoT, it is natural for researchers to use it in practical experiments. Rahman et al. [43], for example, have developed a prototype of the mesh-matrix communication protocol for Local Area Networks (LANs), Personal Area Networks (PANs), Bluetooth, and Near Field Communication (NFC) networks was developed. Implementing the mesh-matrix communication protocol prototype extends the Matrix Protocol functionality to support networks such as LANs, PANs, Bluetooth, and NFC, enhancing its versatility and accessibility across different communication mediums. Brodbeck et al. [35] proposed an educational tool utilizing the Chat App of the Matrix protocol. This educational tool aims to enhance the learning experience by integrating real-time communication features with collaborative tools. Brunila et al. [19] developed the DRIFT, a messaging app that follows a unique communication approach, maintaining time sequence and spatial organization. It utilizes maps to show the message locations. Other papers focus on formal and asymptotic analyses of data replication mechanisms and structuring a Matrix network [3, 11, 12].

Eight papers were assigned the **Only Mentions** category. These papers cover various topics and reference the Matrix protocol. In this way, the Matrix protocol was used as an example of a technology that supported or refuted a particular research feature. Therefore, these mentions can range from briefly comparing technologies to valuable insights. Finally, we believe these papers have contributed significantly to the scientific dissemination of the ecosystem created by the Matrix protocol. Cui et al. [42] have introduced CrossChat enables interaction between users operating on different mobile messaging platforms. It implements a gateway on the device that mediates communication and leverages interface-level interaction techniques to route messages across applications in real time. The method used in CrossChat closely resembles the function of Matrix bridges, which connect different messaging networks, enabling communication across disparate ecosystems. Wichelmann et al. [31] explores the security aspects of the Signal protocol. The Matrix protocol was used to highlight device-specific security code extensions.

RQ02 - Main area of study

Figure 8 shows the papers according to their main area of study. The topic of **Security and Privacy** was addressed in twelve papers, which discuss a wide range of measures to protect user data and ensure secure communication channels. These measures include encryption, which involves encoding data to prevent unauthorized access, authentication processes to verify the identity

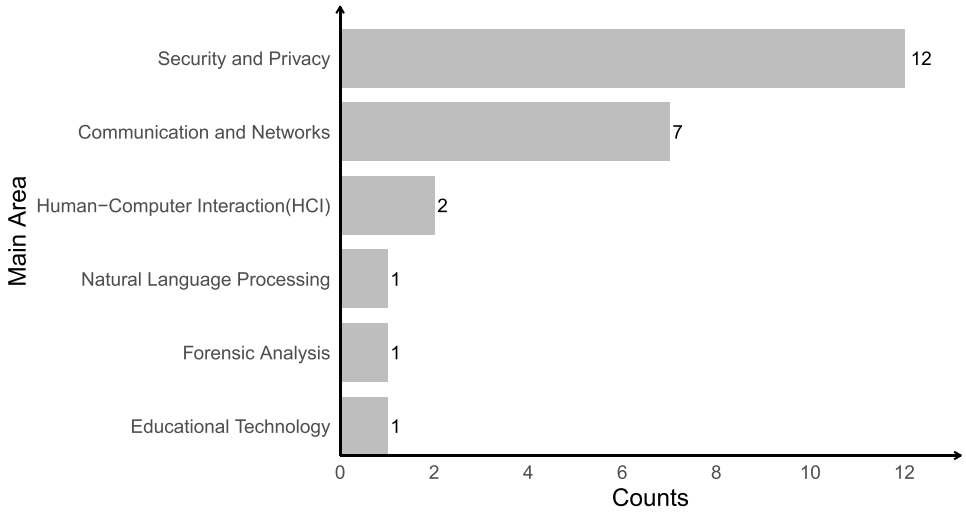


Fig. 8 Primary studies, quantified by main area

of users, and access control mechanisms to manage and restrict user permissions. Most papers addresses **Security and Privacy** category. We believe that the Matrix communication model naturally raises security concerns. In the Matrix federated model, different homeservers host users who can communicate with each other even if they are not connected to the same server. While federation promotes decentralization, it also introduces significant challenges, especially when untrusted servers are involved. Malicious or compromised third-party servers can manipulate messages or metadata, compromising the security and privacy of users. To mitigate these risks, Matrix implements E2EE, ensuring that only end users can access message content. However, potentially malicious servers in the federation are a constant concern as they can harm data integrity or compromise cryptographic key verification. Additionally, Matrix allows users to connect with other communication networks using bridges to interact with external protocols like IRC or XMPP. Despite enhancing interoperability, these bridges introduce new attack vectors, where communication security may be weakened, especially if the target protocol offers a different level of protection than Matrix. Therefore, great interest is expected regarding **Security and Privacy** issues.

Some papers clearly define the threat models used in their experiments and analyses. For instance, Chowdhury et al. [38] paper experiments on six widely used desktop clients using two popular threat elicitation frameworks, STRIDE and LINDDUN.

In our analysis, we assume the presence of a passive third-party adversary. This entity can observe traffic exchanged between clients and servers, attempting to infer message content or communication patterns [19].

In addition, Jacob et al. [11] analyze Byzantine Fault Tolerance and use a simplified threat model for Conflict-Free Replicated Data Type (CRDT). On the other hand, Lu at al. [14] presents a series of discoveries and potential vulnerabilities, including cybersecurity threats in homeservers and compromised confidentiality of encrypted rooms. Other security issues, such as Forward Secrecy, are discussed in the Wichelmann at al. [31], Alwen at al. [39], and Vazquez at al. [32]. Wallez et al. [41] presents a new framework, Comparse, that specifically addresses the security analysis of data formats in cryptographic protocols. Finally, Stajano et al. [44] present a cryptographic challenge-response authentication mechanism to enable portable identities within the Matrix protocol.

In this category, new technologies and approaches are constantly being developed to address emerging challenges. **Networks and Communications** is the second category with the most classified papers. We believe that Matrix's open source license, with example implementations freely licensed without intellectual property restrictions or patent licensing, provides insight and a solid foundation for developing other technologies. This category covers papers that discuss data communication issues, the development of tools and protocols, and network metrics analyses. In some papers, new protocols and tools are proposed. For instance, Rahman at al. [43] introduces Matrix-Mesh, an IM protocol that integrates cloud-dew, client-server, and P2P elements to ensure messaging even without standard connectivity. CrossChat enables communication between different mobile IM apps without modifying the original apps. Cui at al. [42] uses a gateway on mobile devices and UI techniques to capture and exchange messages in real-time. Eisele at al. [34] present TRANSAX, a platform that allows participants to trade energy futures efficiently and privately, helping Distribution System Operators (DSOs). Plan energy needs while ensuring safety while operating in a

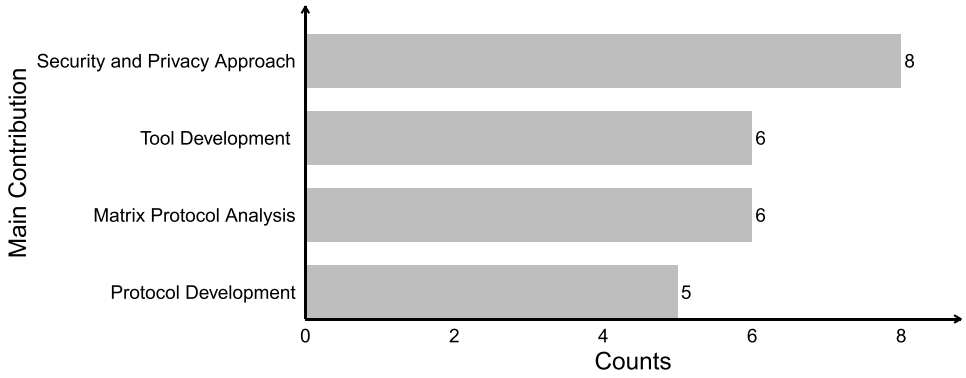


Fig. 9 Primary studies quantified by main contribution

decentralized system provides resilience and must satisfy requirements. Jacob at al. [3] discusses load balancing, asymmetrical communication, and scalability through formal methods analysis or verification of empirical data.

The **Human-Computer Interaction (HCI)** category has two works. Cui at al. [42], extracts the UI tree of IM apps at runtime to avoid modifying the app's software implementation. After extracting the UI tree, CrossChat separates it to identify specific regions for message extraction and sending. Users mark the message window, input box, and send button, which are then used for future operations. This strategy is compared to the use of bridges in the Matrix protocol. In the **Forensic Analysis** category, the author evaluates the Matrix protocol through forensic analysis and provides a detailed report of the communication mechanisms in the form of text, audio, video, and files. The analyses demonstrated that much relevant information can be extracted from the Schipper at al. [2], analyze Matrix protocol and the Riot.im application from a digital forensics perspective. Furthermore, the **Natural Language Processing** category also has one classified article. The author discussed a method for identifying and organizing problem-related information in production chat logs based on employee and company surveys and simulation data to improve digital shop floor management systems. The Natural Language Processing category also explores NLP to enhance interaction between humans and computers. Finally, **Educational Technology** category has one ranked article. The focus is on creating a web-based teaching tool that minimizes bandwidth and power usage by using an event-driven code-casting system instead of video streaming. Brodbeck et al. [35] shows that it supports audio calls and can be extended with modules such as text chat, presentations, Wikipedia, and collaborative text editing. The Matrix chat rooms are used in this tool.

RQ03 - Main contribution

Figure 9 shows the papers according to the main contribution analyzed in the studies.

It is important to note that an article can be classified into multiple categories, reflecting our perception of the work's contribution to a given field of study. Based on these categories, we can observe that most studies focus on the **Security and Privacy Approach** theme, indicating a constant concern with data protection and information integrity. We believe the Matrix Protocol ecosystem requires continuous care to ensure user privacy and promote trust in digital interactions. Operating in decentralized, federated environments while striving for interoperability among network participants presents significant challenges. In decentralized communication networks like the Matrix protocol, security and privacy present unique challenges due to the inherent nature of decentralization. Unlike centralized systems, where a single entity controls data flow, decentralized networks distribute control across multiple independent servers, introducing new vectors for attacks and vulnerabilities. Chowdhury at al. [38], show a critical aspect of securing these networks lies in comprehensive threat modeling that accounts for the diverse attack surfaces presented by federated environments, where varying levels of trust exist between servers. Cryptographic vulnerabilities are a primary concern, as any weaknesses can lead to significant security breaches, such as unauthorized data access or message tampering, compromising the integrity of the entire network. Additionally, in Li et al. [14], privacy considerations are paramount; the decentralized structure makes it challenging to protect user data and communication metadata, which are susceptible to traffic analysis and other privacy-invasive techniques. Federation and trust issues further complicate security efforts, as inconsistencies in security policies across different servers can lead to uneven protection levels, making the network vulnerable to compromised or malicious nodes.

Albrecht et al. [13, 45] argue that addressing security challenges in Matrix requires a robust and evolving cryptographic foundation. Their initial analysis exposed practical weaknesses in key management and authentication workflows, demonstrating that

real-world adversaries could compromise confidentiality and trust in federated deployments. In a subsequent study, the authors expanded this perspective by introducing a formal security model for Matrix, reinforcing the need for continuous evaluation and improvement of cryptographic mechanisms, threat detection strategies, and trust management protocols to ensure secure and privacy-preserving communication in decentralized environments.

Albrecht at al. [13] address these challenges requires a robust and adaptive approach to security, including the continuous evaluation and enhancement of cryptographic methods, threat detection mechanisms, and trust management protocols to safeguard the privacy and security of users in a decentralized environment.

The following papers stand out for their **Tool Development**, each focusing on essential secure and decentralized communication aspects. These four papers demonstrate a significant convergence in developing secure and decentralized communication tools, emphasizing interoperability, efficiency, and privacy. Cui et al. [42] and Brodbeck and Hromada [35] address the need for efficient communication in various contexts, whether through interoperability between messaging apps or by reducing bandwidth consumption in online teaching environments. Both focus on accessibility and adaptability to different platforms and connectivity conditions. Wallez et al. [41], in turn, provides a solid foundation for the security of these communications, ensuring that the cryptographic formats used are verifiable and secure, a central concern also addressed in Eisele et al. [34], which focuses on privacy and transparency in decentralized financial transactions. Together, these contributions highlight the importance of tools that facilitate communication and interaction between diverse systems and ensure users' security and privacy in increasingly decentralized and interconnected environments. Moreover, all the papers demonstrate a concern for balancing technological innovation with simplicity and usability, ensuring that the developed tools are practical and adaptable to the real needs of users. The emphasis on decentralization in these tools reflects a growing trend to move away from centralized systems, vulnerable to failures and attacks, and towards promoting a more resilient and robust architecture. Finally, by integrating aspects of interoperability, security, and efficiency, these tools present new approaches to immediate technical challenges and contribute to advancing secure communication and transaction practices in a rapidly evolving digital world.

In the process of **Protocol Development**, the papers of Rahman et al. [43], Alwen et al. [39], Vazquez et al. [32], and Weidner et al. [9] converge on highlighting security and resilience as crucial components, particularly in decentralized contexts. For instance, Alwen et al. [39] and Weidner et al. [9] explore the use of cryptographic key agreements, such as Elliptic Curve Diffie-Hellman (ECDH), to ensure that communications between parties are secure and authenticated, even in environments where trust among nodes is limited. Additionally, Rahman et al. [43] and Vazquez et al. [32] emphasize the need to design protocols that are not only secure but also efficient and scalable. Rhman et al. [43], for example, proposes a protocol that can adapt to various network conditions, including scenarios with limited connectivity. At the same time, Vazquez et al. [32] addresses lightweight and scalable authentication methods, such as PAKE (Password Authenticated Key Exchange), to prevent system or user overload. All the papers underscore the importance of adopting recognized cryptographic standards, such as the Advanced Encryption Standard (AES) and SHA-256, to ensure the interoperability and reliability of the developed solutions. These studies thus point to a development approach that balances innovation with decentralized systems' stringent security, efficiency, and scalability requirements.

While addressing different aspects of the Matrix protocol, the category **Matrix Protocol Analysis** converges on the importance of analyzing scalability, federation, and interoperability as fundamental pillars for the network's success and security. As seen in Jacob et al. [12] and Jacob et al. [11], the scalability analysis is essential to ensure that Matrix can grow without sacrificing performance or security. Federation, discussed in detail in Franco et al. [46] and Li et al. [14], offers both benefits and challenges, providing resilience through decentralization but also introducing complexity in synchronization and maintaining network cohesion. Interoperability is another critical aspect, enabling different servers and implementations to function seamlessly, as highlighted in Li et al. [14] and Jacob et al. [3]. Finally, the cryptographic vulnerabilities explored in Albrecht et al. [13] demonstrate that despite the advantages offered by federation and interoperability, it is vital that all servers adhere to strict security protocols to avoid compromises. Together, these papers provide a holistic view that connects the technical architecture of Matrix with the practical challenges of maintaining a secure, scalable, and interoperable network in a decentralized environment.

RQ04 - Explored Matrix feature

Figure 10 shows the papers according to the characteristics of the Matrix protocol explored by the papers. Accordingly, we classify the papers' approach into the following characteristics: Group Communication, Security Issues, and Matrix Architecture. The **Group Communication** category encompasses papers that discuss the message exchange mechanisms of the Matrix protocol, addressing issues related to the creation of homeservers,

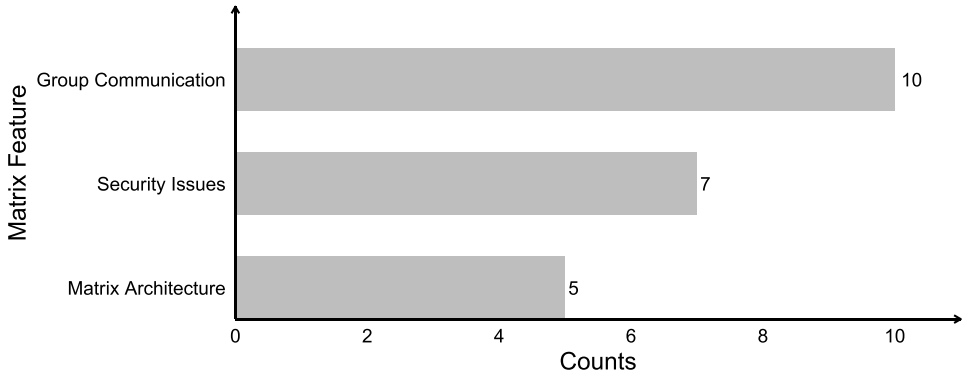


Fig. 10 Primary studies quantified by explored Matrix feature

users, and groups. The Security Issues category classifies papers that address security and privacy issues in the Matrix protocol, such as E2EE and user authentication. On the other hand, the Matrix Architecture category includes studies that analyze the structure and functioning of the Matrix protocol as a whole, exploring aspects related to both Group Communications and Security Issues, among other concerns.

The Group Communication category was the most frequently used to classify works in this research question. We believe this is because a large portion of the paper addresses issues related to federation, such as interoperability and scalability. Additionally, many papers showed great interest in using user chat rooms as a communication model in developing their applications. Jacob et al. [12] directly explore the scalability and centralization issues in server-to-server communication, identifying how the federated structure of Matrix can impact the efficiency of message exchange in groups. In parallel, Brunila et al. [19] and Gibb et al. [40] contribute to the discussion by examining secure group communication, emphasizing E2EE and the application of Matrix in geospatial contexts. Brodbeck and Hromada [35] complement this analysis by adapting the protocol for low-bandwidth environments, ensuring the continuity of group communication under adverse conditions. Cui et al. [42] address interoperability, demonstrating how Matrix can connect different instant messaging platforms, expanding the possibilities for communication between servers of distinct networks. Though focused on extracting problem-related entities in production environments, Müller et al. [33] provides relevant insights into message management and the effectiveness of group communication within Matrix. These studies explore the multiple facets of Group Communication in a Matrix of decentralized communication. When analyzed collectively, these studies offer an exploration of the various dimensions of Group Communication within the Matrix protocol, addressing everything from technical challenges in scalability and server-to-server communication efficiency to critical security and platform interoperability issues. By investigating how Matrix handles group communication in different contexts—whether adapting to low-bandwidth environments, implementing E2EE, or integrating multiple messaging systems—these studies not only elucidate the protocol's strengths and limitations but also pave the way for the development of more robust and scalable solutions. These contributions are important for enhancing decentralized communication, ensuring that Matrix can be effectively applied across a broad spectrum of use cases.

The analyzed papers provide a comprehensive view of **Security Issues** in the Matrix protocol, addressing everything from device compromise protection to the integrity of communications in distributed environments. Wichelmann et al. [31] explore vulnerabilities associated with compromised devices, highlighting the importance of post-compromise security to protect communications within Matrix. Weidner et al. [9] examine decentralized key agreements for secure messaging, which is essential for maintaining security in unstable networks. Chowdhury et al. [38] provide a detailed analysis of threat models in distributed networks, underscoring the need for robust encryption to mitigate attacks. Alwen et al. [39] introduce a fork-resilient protocol, crucial for ensuring the security and integrity of group communications in Matrix. Wallez et al. [41] address the security of cryptographic formats, preventing attacks that could compromise message integrity. Schipper et al. [2] complement this discussion by exploring the forensic analysis of Matrix and the Riot.im client, emphasizing the security challenges related to the acquisition and analysis of encrypted data. Finally, Brunila et al. [19] focus on secure instant messaging with end-to-end encryption, emphasizing the importance of privacy in Matrix communications. Together, these papers lay a foundation for enhancing security practices in Matrix, ensuring more secure and resilient communication in decentralized systems.

The papers investigating **Matrix Architecture** provides a detailed analysis of the structural and functional elements of the Matrix protocol, addressing key concepts such as Event-Based Communication, Client-Server Model, and Federation. Rahman et al. [43] explore the implementation of a hybrid protocol that combines the Client-Server Model with P2P, emphasizing the flexibility required to operate under limited connectivity conditions, where Federation plays a critical role. Li et al. [14] demystify decentralized networks based on Matrix, demonstrating how Federation and Event-Based Communication ensure the scalability and security of server-to-server communication. Albrecht et al. [13] highlight cryptographic vulnerabilities within the Client-Server Model, underscoring the importance of a robust architecture to prevent security breaches. Jacob et al. [11] and Jacob et al. [3] investigate Event-Based Communication and Federation, revealing how these features ensure consistency and resilience in distributed and federated environments. Together, these studies explain how Matrix architecture integrates these elements to create a secure, scalable, and efficient platform capable of operating in various complex scenarios. In addition to explaining these issues, the papers also highlight weaknesses within the Matrix ecosystem, such as the need for improved formal verification and the potential exploitation of cryptographic vulnerabilities, pointing to critical areas for future improvement.

Research challenges and directions

The adoption of the Matrix protocol in decentralized communication systems presents both opportunities and challenges, particularly in ensuring seamless integration and optimized performance across various domains. As the protocol continues to gain traction, several critical challenges emerge, demanding focused research and innovative solutions. These difficulties are a result of the decentralized architectures' inherent complexity, the necessity of strong security and privacy safeguards, the high performance demands of real-time communication, and the need to scale efficiently while preserving interoperability across various platforms and applications. Understanding these challenges is essential for researchers and developers aiming to enhance the Matrix protocol's implementation in real-world environments. In this section, we explore the key research areas that need attention, highlighting existing barriers and potential directions for future studies. By addressing the pressing concerns of security and privacy, optimizing real-time communication processes, tackling scalability and performance issues, and ensuring interoperability, this study aims to provide a foundation for further advancements in decentralized communication technologies. The following subsections present each of these critical areas, offering insights into current research gaps and the directions in which the Matrix protocol's development can be steered to foster broader adoption and integration.

Security and privacy

Cryptographic vulnerabilities: Cryptographic vulnerabilities present a significant challenge in the Matrix protocol, especially when dealing with weak authentication methods, susceptibility to MITM attacks, and protocol confusion issues. Weak authentication can lead to unauthorized access, allowing malicious actors to impersonate legitimate users or systems, thus compromising communication security. Albrecht et al. [13] analyzed attacks like MITM. The susceptibility to MITM attacks is particularly concerning in decentralized networks like Matrix, where the flow of information between servers and clients can be intercepted if encryption is not properly implemented. In these cases, attackers can manipulate or eavesdrop on the communication without the knowledge of the communicating parties. Wichelmann et al. [31] discuss protocol confusion, where different implementations of a protocol handle messages inconsistently, also introduces risks, as attackers may exploit these discrepancies to create vulnerabilities or bypass security measures. These vulnerabilities highlight the need for stronger cryptographic mechanisms and standardized practices across the Matrix ecosystem to ensure robust security. Without addressing these issues, the protocol remains vulnerable to attacks that can undermine its decentralized and secure communication objectives.

Key compromise by malicious servers: According to Chouwdhury et al. [38] in the decentralized architecture of the Matrix protocol, key compromise by malicious servers poses a particularly insidious threat. The federated nature of Matrix means that different servers can communicate and share information, but this also means that a rogue or compromised server could manipulate cryptographic keys to decrypt sensitive messages or even inject malicious content into conversations. This becomes a serious issue when E2EE is in place, as the server could potentially exploit flaws in key distribution or management to gain unauthorized access to private communication. Additionally, a compromised server could disrupt the integrity of key exchange protocols, injecting malicious keys or manipulating sessions to allow an attacker to eavesdrop on or alter communications. As noted by Jacob et al. [3], the lack of a centralized trust authority makes it difficult to prevent such compromises across federated networks, making it essential to develop more robust server verification techniques and key management strategies to mitigate these risks. Without these safeguards, malicious servers can exploit the very openness of Matrix's federated design.

Lack of formal verification in access control: The absence of formal verification in access control

mechanisms is another pressing issue in the Matrix protocol, as it leaves significant room for vulnerabilities and misconfigurations that could be exploited by malicious actors. As observed by Jacob et al. [3], formal verification provides a mathematically rigorous method of proving that access control systems function as intended, ensuring that only authorized users have access to certain resources or communications. Without this layer of assurance, Matrix implementations may contain logic errors or inconsistencies in how access is granted or denied, potentially allowing unauthorized users to gain access to sensitive communications or system resources. This problem is exacerbated by the decentralized nature of Matrix, where different servers and clients may implement access control in slightly different ways, leading to inconsistencies in enforcement. As mentioned in Jacob et al. [11], the lack of formal verification makes it difficult to guarantee that all nodes within a federated system adhere to the same security principles, thereby introducing gaps in security that could be exploited by attackers. Addressing this issue would require developing standardized, formally verified access control mechanisms that can be consistently applied across the entire Matrix ecosystem.

Lack of a unified security model: One of the most significant challenges in the Matrix protocol is the lack of a unified security model across federated servers. In a federated system, different servers are responsible for maintaining their own security policies, which can vary widely in terms of robustness and implementation. This disparity can be exploited by attackers who may find and target weaker servers in the network, thereby compromising the security of the entire federated system. Without a unified security model, it is difficult to ensure consistent protection across all servers, and this inconsistency can lead to security gaps where malicious actors can infiltrate the network. A unified security model would provide a set of standardized security guidelines and protocols that all servers must follow, ensuring a baseline level of security throughout the federated network. The current lack of such a model in Matrix means that security can vary significantly between servers, creating a situation where the overall security of the protocol is only as strong as its weakest link. Standardizing security policies across the federated network is crucial for maintaining the integrity and trustworthiness of the protocol.

Key management issues: Key management is a critical issue in the Matrix protocol, given the use of multiple cryptographic keys for different aspects of communication, such as E2EE, server communication, and identity verification. Managing these keys across a decentralized and federated system introduces significant complexity, as each server and client must maintain and securely store a wide range of keys. The complexity of managing multiple keys increases the risk of key compromise, mismanagement, or loss, any of which could have catastrophic consequences for the security of communications. Furthermore, decentralized key management lacks the centralized oversight that can help ensure keys are handled consistently and securely across the entire system. This makes it difficult to enforce best practices for key rotation, storage, and disposal, which are essential for maintaining long-term security. The current approach to key management in Matrix leaves room for improvement, particularly in terms of developing automated systems that can handle key generation, storage, and rotation in a secure and efficient manner, reducing the burden on users and system administrators while improving overall security.

Potential weaknesses in End-to-end encryption: E2EE is a core feature of the Matrix protocol. However, it does have some weaknesses. The implementation or configuration of encryption algorithms can introduce vulnerabilities that could be exploited by attackers. For instance, if encryption algorithms are not implemented according to best practices or if configuration errors occur during setup, it could leave communications vulnerable to interception or decryption. Additionally, the decentralized nature of Matrix adds further complexity to ensuring that E2EE is consistently applied across all communication channels and devices. Weaknesses in the implementation of E2EE could arise from differences in how various clients or servers handle encryption, leading to situations where certain communications are not as secure as intended. This can be particularly problematic in federated networks, where not all participants may adhere to the same security standards. Addressing potential weaknesses in E2EE requires ongoing audits, standardization of encryption practices, and rigorous testing to ensure that all elements of the Matrix ecosystem are capable of providing strong, consistent encryption for all communications.

Real-time communication

Message latency and synchronization: Message latency is a critical issue in real-time communication, and the decentralized communication model of the Matrix protocol exacerbates these challenges. Unlike centralized systems, where messages pass through a single server, the Matrix protocol involves multiple servers in a federated network, each potentially located in different regions or under varying network conditions. This can lead to significant delays in message delivery, as each server must process and forward messages independently. Latency can be further compounded by the need for synchronization between different servers, which may not always be perfectly aligned due to differences in processing speeds or network conditions. This lack of consistent synchronization can cause messages to be delayed or delivered

out of order, impacting the user experience, especially in environments where timely communication is critical, such as collaborative workspaces or chat systems. Minimizing message latency requires improvements in both server processing capabilities and network efficiency, as well as the development of better synchronization mechanisms to ensure that messages are delivered promptly and in the correct order across all servers.

Maintaining message order: The decentralized nature of the Matrix protocol poses unique challenges in maintaining message order, particularly when dealing with variable network latencies. In real-time communication, ensuring that messages are displayed in the correct order is crucial for preserving the logical flow of conversations. However, in a federated system like Matrix, messages may take different paths through the network, with each path introducing varying amounts of latency. As a result, it is possible for messages sent in rapid succession to arrive out of order, which can cause confusion and disrupt the flow of communication. To address this, the Matrix protocol must rely on mechanisms for reordering messages based on timestamps or other metadata, ensuring that the final display of messages reflects their intended order. This requires careful coordination between servers to track message sequences and resolve any discrepancies caused by network delays. Maintaining message order in decentralized environments remains an ongoing challenge that necessitates continuous refinement of synchronization and latency mitigation techniques.

Quality of Service (QoS) assurance: Ensuring consistent QoS in real-time communication is essential to minimizing delays, jitter, and packet loss, all of which can significantly degrade the user experience. QoS is particularly challenging in decentralized systems like Matrix, where the performance of communication is dependent on a wide variety of factors, including the network conditions of each participating server, the geographic distribution of those servers, and the overall load on the system. Without adequate QoS measures in place, users may experience interruptions in their communication, with messages being delayed or lost entirely. To address this, the Matrix protocol must incorporate mechanisms for monitoring network conditions in real time and adjusting traffic priorities to minimize the impact of congestion or packet loss. Additionally, jitter – the variation in packet arrival times – must be controlled to ensure smooth communication, particularly in scenarios involving multimedia, such as video conferencing or voice calls. Developing more robust QoS frameworks for the Matrix protocol is essential to ensuring reliable real-time communication, regardless of the underlying network conditions.

Integration of Voice over IP (VoIP): The integration of Voice over IP (VoIP) into the Matrix protocol presents a unique set of challenges, particularly with respect to synchronization and latency issues. In real-time voice communication, even small delays or interruptions can result in a poor user experience, as participants may speak over one another or experience gaps in conversation. Latency becomes especially problematic when servers are spread across different regions, as the time it takes for voice packets to traverse the network can introduce noticeable delays. In addition, VoIP requires precise synchronization between servers to ensure that audio streams are transmitted in real time without noticeable interruptions or loss of quality. If synchronization is not properly maintained, voice calls may suffer from echoes, gaps, or other quality issues that make communication difficult. Addressing these challenges requires both optimizing the underlying network infrastructure for real-time audio transmission and developing more sophisticated algorithms for handling synchronization and latency in decentralized environments like Matrix.

Synchronization conflict resolution: Synchronization conflicts arise in the Matrix protocol when multiple servers attempt to process events simultaneously, leading to inconsistencies in the order or content of messages. In real-time communication, these conflicts can be particularly disruptive, as they may result in messages being displayed out of order or entirely missed by one or more participants. Such conflicts are inherent to decentralized systems, where different servers may not always be perfectly synchronized due to varying network conditions or processing speeds. To resolve these conflicts, the Matrix protocol must implement mechanisms for detecting and reconciling discrepancies between servers, ensuring that all participants receive a consistent view of the conversation. This may involve the use of timestamps, sequence numbers, or other metadata to track the order of events and resolve conflicts when they arise. Effective conflict resolution is critical to maintaining the integrity of real-time communication in Matrix, as even small inconsistencies can lead to significant disruptions in the user experience.

Network connection and firewall management: Real-time communication in the Matrix protocol is further complicated by the need to manage network connections across various environments, including traversing firewalls and Network Address Translation (NAT) systems. Firewalls and NATs are commonly used to enhance security by restricting access to certain ports or IP addresses, but they can also introduce challenges for decentralized communication protocols like Matrix, which require seamless peer-to-peer connections. Establishing and maintaining these connections can be difficult when firewalls block necessary traffic or when NAT devices reassign internal IP addresses, disrupting communication. To overcome these obstacles, the Matrix protocol must implement techniques such as hole punching, relays, or

TURN servers, which can help establish connections through firewalls and NATs without compromising security. However, these solutions can introduce additional latency and complexity, making it crucial to balance security and performance in real-time communication scenarios. Effectively managing network connections and firewall traversal is essential to ensuring that Matrix can provide reliable and efficient real-time communication, even in complex network environments.

Scalability and performance

Scalability limitations in federated servers: As the number of federated homeservers in the Matrix protocol grows, scalability limitations become more pronounced, particularly in terms of data synchronization. Federation allows different servers to communicate and share information, but as the number of servers increases, so does the complexity of maintaining consistent data across them. This complexity stems from the need to continuously synchronize messages, events, and user states between servers, which can become increasingly challenging as the network expands. With each additional server, the amount of data that must be processed, stored, and synchronized increases, placing more strain on server resources. Moreover, the decentralized nature of Matrix means that there is no central authority to enforce synchronization standards, making it difficult to ensure that all servers remain up-to-date with the latest data. As a result, scalability becomes a key challenge for the Matrix protocol, requiring improvements in both the underlying architecture and the algorithms used for data synchronization to accommodate the growing number of federated servers without sacrificing performance or consistency.

Impact of data volume on performance: The volume of data generated by the Matrix protocol can have a significant impact on server performance, particularly as the number of events and messages in the network grows. Each message or event must be processed, stored, and synchronized across multiple servers, and as the amount of data increases, so does the load on each server. This can lead to performance degradation, as servers struggle to keep up with the demands of processing large volumes of data in real-time. In particular, chat rooms with high levels of activity or a large number of participants can generate a substantial volume of events, leading to server overload and delays in message delivery. Additionally, the need to store and retrieve historical messages and events adds further strain on server resources, particularly in cases where large amounts of data must be accessed quickly. To mitigate the impact of data volume on performance, the Matrix protocol must implement more efficient data processing and storage mechanisms, as well as optimize the algorithms used for event synchronization and retrieval.

Event synchronization between servers: Event synchronization between federated servers is a fundamental aspect of the Matrix protocol, but it also introduces challenges in terms of performance and consistency. As events are generated in a chat room or communication channel, they must be synchronized across all participating servers to ensure that users receive consistent updates in real-time. However, this continuous synchronization process can cause delays, particularly when there are network latencies or variations in server processing speeds. In some cases, synchronization issues can lead to inconsistencies in message delivery, with certain servers receiving events out of order or experiencing delays in receiving updates. This can disrupt the flow of communication and degrade the user experience, especially in real-time communication scenarios where timely message delivery is critical. Improving event synchronization between servers is essential to enhancing the performance and reliability of the Matrix protocol, particularly as the network grows and the volume of events increases.

Event processing overload: Event processing overload is a common issue in the Matrix protocol, particularly in chat rooms with a large number of participants. As each participant sends messages or generates events, the server must process and synchronize these events with other federated servers, which can lead to delays if the volume of events exceeds the server's processing capacity. This overload can result in delays in message delivery, as servers struggle to keep up with the demands of real-time communication. Additionally, chat rooms with high levels of activity can generate a significant amount of metadata, such as read receipts and typing indicators, which further adds to the processing load. To address event processing overload, the Matrix protocol must implement more efficient event processing algorithms, as well as optimize server resources to handle large volumes of events without causing delays or disruptions in message delivery. By improving event processing efficiency, the Matrix protocol can better accommodate chat rooms with large numbers of participants and high levels of activity.

Delays in message propagation in federated networks: Message propagation delays are a significant challenge in federated networks, such as those used in the Matrix protocol. As messages are sent between servers, they must traverse the network, which can introduce delays due to both network latency and the time required for servers to process and forward messages. In a federated system, each server operates independently, and variations in network conditions or server processing speeds can lead to uneven message delivery times. This can result in certain participants receiving messages later than others,

disrupting the flow of conversation and potentially causing confusion. Additionally, the decentralized nature of Matrix means that there is no central authority to ensure consistent message propagation, making it difficult to guarantee timely message delivery across all servers. To minimize message propagation delays, the Matrix protocol must optimize its routing algorithms and improve network efficiency, ensuring that messages are delivered as quickly as possible, regardless of the underlying network conditions.

Performance of cryptographic processing at scale: The use of E2EE in the Matrix protocol adds a layer of security to communications, but it also introduces significant computational overhead, particularly at scale. E2EE requires that each message be encrypted and decrypted by the sender and recipient, respectively, and as the volume of messages increases, so does the demand for cryptographic processing. This can result in performance issues and high message throughput, as the computational cost of encryption and decryption may become a bottleneck. Furthermore, the decentralized nature of Matrix means that each server must handle its own cryptographic processing, which can lead to inconsistencies in performance across the network. To address the performance challenges associated with cryptographic processing at scale, the Matrix protocol must implement more efficient encryption algorithms and optimize the cryptographic processes used in message transmission. By reducing the computational overhead of E2EE, the Matrix protocol can maintain its security features without sacrificing performance in large-scale environments.

Interoperability

Backward compatibility and protocol complexity: Maintaining backward compatibility in the Matrix protocol is a significant challenge, given the increasing complexity of the protocol as new versions and features are introduced. In a decentralized and federated system like Matrix, where multiple servers and clients interact, ensuring that different versions of the protocol can coexist without disrupting communication is crucial. However, this becomes difficult as older implementations may not support newer features or security improvements, leading to compatibility issues that can affect the overall user experience. The need to maintain compatibility between different versions also increases the complexity of the protocol, as developers must ensure that new changes do not inadvertently break existing functionality. This can lead to situations where certain features are either disabled or only partially supported in older versions, creating a fragmented user experience. Addressing backward compatibility requires careful planning and design, as well as the development of standardized practices to ensure that updates to the protocol can be rolled out smoothly without compromising the stability and usability of the system.

Event synchronization and scalability: Event synchronization is a critical aspect of interoperability in the Matrix protocol, particularly as the network scales. As messages and events are generated, they must be synchronized across all participating servers to ensure that users receive consistent updates in real-time. However, as the network grows and the number of servers increases, synchronization becomes more complex, leading to potential delays in message delivery or partial synchronization. This can result in situations where certain users receive messages later than others, or where messages are lost or duplicated due to incomplete synchronization. Ensuring that event synchronization remains efficient and reliable as the network scales is essential for maintaining the integrity of the communication system. This requires the development of more robust synchronization algorithms and improvements in the underlying infrastructure to handle the increased load without sacrificing performance or reliability.

Federation with non-Matrix systems: Interoperability with non-Matrix systems presents a unique set of challenges, particularly when it comes to federation between different communication platforms. Matrix is designed to be an open and federated protocol, but when interacting with systems that use different data standards or encryption methods, inconsistencies can arise. These differences can lead to data loss, message duplication, or other issues that disrupt the flow of communication. For example, non-Matrix systems may not support the same level of encryption, resulting in messages that are either unencrypted or improperly secured when exchanged between platforms. Additionally, variations in data formats and messaging standards can lead to difficulties in parsing or displaying messages, further complicating cross-platform communication. To address these challenges, Matrix must implement more flexible federation mechanisms that can adapt to the specific requirements of non-Matrix systems while ensuring that security and data integrity are maintained.

Byzantine failures and synchronization guarantees: Byzantine failures present a significant challenge in decentralized systems like Matrix, particularly when it comes to ensuring reliable synchronization across servers. Byzantine failures occur when a server behaves arbitrarily or maliciously, potentially sending incorrect or inconsistent messages to other servers in the network. In the Matrix protocol, which relies on "best-effort" transmission mechanisms, there are limited guarantees that messages will be delivered consistently or in the correct order, especially in the face of Byzantine failures. This can lead to situations where certain users experience delays, receive corrupted messages, or miss messages entirely

due to synchronization issues. To mitigate the impact of Byzantine failures, Matrix must implement stronger synchronization guarantees and fault-tolerant mechanisms that can detect and recover from these failures, ensuring that communication remains reliable even in the presence of malicious or faulty servers. Without these safeguards, the decentralized nature of Matrix leaves the protocol vulnerable to inconsistencies and disruptions in communication.

Identity management and user discovery: User discovery and identity management are critical components of interoperability in the Matrix protocol, but they pose significant challenges, particularly when dealing with users across different domains. Matrix relies on identity servers to help users discover each other, but the decentralized nature of the protocol makes it difficult to maintain a consistent and reliable system for managing identities. Different domains may use different identity servers, and users may have multiple identities across different platforms, leading to confusion and difficulties in discovering and connecting with other users. Additionally, the lack of a unified identity management system can create security vulnerabilities, as attackers may exploit inconsistencies in the way identities are verified or managed across servers. To address these challenges, Matrix must develop more standardized identity management practices and improve user discovery mechanisms to ensure that users can reliably find and connect with each other, regardless of the domain or identity server they are using.

Implementation differences in clients and servers: Interoperability in Matrix is also affected by differences in the implementation of clients and servers, with certain functions or APIs not uniformly supported across the ecosystem. As Matrix is an open protocol, different developers may create their own clients and servers, each with slightly different implementations of the protocol. This can lead to inconsistencies in the way certain features are supported or implemented, resulting in fragmented functionality. For example, a particular API might be fully supported on one server but only partially implemented on another, leading to differences in the user experience depending on which client or server is being used. These implementation differences can also affect compatibility between servers, as certain features may not work as expected when interacting with servers that do not fully support the protocol. To improve interoperability, Matrix must work towards standardizing the implementation of its protocol across different clients and servers, ensuring that key features and APIs are consistently supported throughout the ecosystem.

Conclusion

The Matrix protocol has emerged as a vital technology in decentralized communication, offering significant advantages in terms of security, scalability, and interoperability. This systematic mapping study has provided a comprehensive overview of the current research landscape surrounding the protocol, categorizing studies and analyzing key contributions related to its adoption. Through the examination of 156 papers, narrowed down to 21 primary studies, this research has highlighted trends, research gaps, and opportunities for further exploration. One of the primary findings of this study is that while the Matrix protocol shows great promise in fostering secure, real-time communication across federated servers, several critical challenges remain. Issues related to scalability, event synchronization, and cryptographic overhead need to be addressed to improve performance, particularly in large-scale and high-traffic environments. Moreover, interoperability with non-Matrix systems remains a complex task, requiring more advanced mechanisms to bridge the differences in data standards and encryption protocols.

Security is another major area of focus, as demonstrated by the frequent discussion of E2EE and key management challenges in the reviewed literature. The protocol's reliance on decentralized servers creates opportunities for malicious actors to exploit vulnerabilities, particularly in key compromise scenarios. Strengthening the cryptographic backbone of Matrix and ensuring robust key management across federated servers will be essential for future developments. Interoperability and the need for a unified security model across federated systems are crucial factors that will influence the future adoption of Matrix. The protocol's ability to support various communication platforms highlights its flexibility, but this flexibility must be balanced with consistent security practices and performance standards. The lack of a unified security model across federated servers is an ongoing concern, one that requires urgent attention to prevent inconsistencies and vulnerabilities across the network.

This study also underscores the need for more structured evaluation frameworks to measure the protocol's effectiveness in real-world applications. By identifying research gaps, such as the limited focus on formal verification methods and scalability solutions, this paper provides a roadmap for future research to improve the Matrix protocol's robustness and reliability.

Future research on the Matrix protocol should focus on several key areas. First, formal verification methods for access control and security models need to be developed to enhance resilience against attacks. Scalability improvements are crucial, especially in optimizing event synchronization and message propagation for large-scale

deployments. Additionally, advanced solutions for cross-platform interoperability, particularly in bridging Matrix with non-Matrix systems, must be explored to maintain security and data integrity. Reducing cryptographic overhead while maintaining E2EE is another priority to improve performance. Lastly, enhancing user experience and identity management, including simplifying user discovery and verification, will support broader adoption of the protocol.

Authors' contributions
The authors contributed equally to this work.

Funding
No funding was received for conducting this study.

Data availability
No datasets were generated or analysed during the current study.

Declarations

Competing interests
The authors declare no competing interests.

Received: 4 December 2024 / Accepted: 7 December 2025
Published online: 18 February 2026

References

1. Matrix F (2024) An open network for secure, decentralised communication. https://matrix.org/. Accessed: 2024-08-25
2. Schipper GC, Seelt R, Le-Khac NA (2021) Forensic analysis of Matrix protocol and Riot.Im application. Forensic Sci Int: Digit Investigation 36:301118. https://doi.org/10.1016/j.fsidi.2021.301118
3. Jacob F, Becker L, Grashöfer J, Hartenstein H (2020) Matrix decomposition: analysis of an access control approach on transaction-based DAGs without finality. In Proceedings of the 25th ACM Symposium on Access Control Models and Technologies, SACMAT'20, Association for Computing Machinery. event-place, New York, NY, USA,: Barcelona, Spain., 81–92
4. Alatawi M, Saxena N (2023) SoK: an analysis of end-to-end encryption and authentication ceremonies in secure messaging systems. In Proceedings of the 16th ACM Conference on Security and Privacy in Wireless and Mobile Networks, WiSec'23, New York, NY, USA,: Guildford, United Kingdom., 187–201. Association for Computing Machinery. event-place
5. Foundation M (2022) Mozilla's use of matrix for internal communications. https://blog.mozilla.org/. Accessed: 2024-08-25
6. E (2024) Element: secure messaging app based on matrix protocol. https://element.io/. Accessed: 2024-06-08
7. NHS Digital (2023) How the nhs uses matrix for secure communications. https://digital.nhs.uk/matrix/. Accessed: 2024-10-07
8. Germany G (2022) Berlin's digital transformation: integrating secure communication with matrix protocol. https://govtech.de/berlim-matrix/. Accessed: 2024-10-07
9. Weidner M, Kleppmann M, Hugenroth D, Beresford AR (2021) Key Agreement for decentralized secure group messaging with strong security guarantees. In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, CCS'21, New York, NY, USA, 2024–2045. Association for Computing Machinery. event-place: Virtual Event, Republic of Korea
10. Government F (2020) Tchap: the official messaging platform for French civil servants. https://www.tchap.gouv.fr/. Accessed: 2024-08-25
11. Jacob F, Beer C, Henze N, Hartenstein H (2021) Analysis of the Matrix event Graph Replicated data type. IEEE Access 9:28317–28333. https://doi.org/10.1109/ACCESS.2021.3058576
12. Jacob F, Grashöfer J, Hartenstein H (2019 December) A glimpse of the Matrix: scalability issues of a new message-oriented data synchronization middleware. In Proceedings of the 20th International Middleware Conference Demos and Posters, Middleware'19, Association for Computing Machinery, New York, NY, USA, 5–6
13. Albrecht MR, Celi S, Dowling B, Jones D (2023 May) Practically-exploitable cryptographic vulnerabilities in Matrix. In 2023 IEEE Symposium on Security and Privacy (SP), pp 164–181.ISSN:2375-1207
14. Li H, Wu Y, Huang R, Mi X, Hu C, Guo S (2023 December) Demystifying decentralized Matrix communication network: ecosystem and security. In 2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS), pp 260–267. ISSN:2690-5965
15. Project F Fluffychat: a matrix client with a focus on user-friendly design. https://fluffychat.im/. Accessed: 2024-06-08
16. Project H Hydrogen: a lightweight matrix client for web and mobile. https://hydrogen.element.io/. Accessed: 2024-06-08
17. Bernstein DJ (2006) Curve25519: new diffie-hellman speed records. https://cr.yp.to/ecdh/curve25519-20060209.pdf. Accessed: 2024-08-01
18. Moerel L, Timmers P (2021) Digital sovereignty and the open strategic autonomy of europe. SSRN Electron J 10(2139/ssrn.3776179)
19. Brunila M, McConnell M, Grigg S, Appuhn M, Sumner B, Bohman M (2022) Drift: E2EE Spatial feature sharing & instant messaging. In Proceedings of the 6th ACM SIGSPATIAL International Workshop on Location-Based Recommendations, Geosocial Networks and Geoadvertising, LocalRec'22, New York, NY, USA.: Seattle, Washington. Association for Computing Machinery. event-place
20. Hoffman P, McGrew D, Kampanakis P (2022) Transitioning to post-quantum cryptography. Internet Eng Task Force (IETF) Draft IETF draft-ietf-pqc-transition-05
21. Of Standards NI, Technology (2022) Post-quantum cryptography: NIST's plan for the future. Technical report, NIST. Accessed April 2025
22. Project S Schildichat: a matrix client with enhanced user experience features. https://schildi.chat/. Accessed: 2024-06-08
23. Foundation M Synapse: matrix reference homeserver. https://github.com/matrix-org/synapse. Accessed: 2024-06-08
24. Foundation M Dendrite: next-generation matrix homeserver written in go. https://github.com/matrix-org/dendrite. Accessed: 2024-06-08
25. Project C Conduit: lightweight matrix homeserver written in rust. https://conduit.rs/. Accessed: 2024-06-08
26. Johansen C, Mujaj A, Arshad H, Noll J (2021 July) The snowden phone: a comparative Survey of secure instant messaging mobile applications. Secur And Commun Networks 2021:1–30. https://doi.org/10.1155/2021/9965573
27. Blessing J (2023) One protocol to rule them all? On securing interoperable messaging (transcript of discussion). In: Security protocols XXVIII: 28th international workshop. Springer-Verlag. event-place, *Cambridge, UK*, Berlin, Heidelberg: Cambridge, United Kingdom, pp 193–198. *March 27–28*, 2023,Revised Selected Papers
28. Blessing J, Chowdhury PD, Sameen M, Anderson R, Gardiner J, Rashid A (2023) Towards Human-centric endpoint security. In: Stajano F, Matyáš V, Christianson B, Anderson J (eds) Security protocols XXVIII. Springer Nature Switzerland, Cham, pp 211–219
29. Kitchenham B, Charters S et al. (2007) Guidelines for performing systematic literature reviews in software engineering
30. Petersen K, Vakkalanka S, Kuzniarz L (2015) Guidelines for conducting systematic mapping studies in software engineering: an update. Inf And Softw Technol 64:1–18
31. Wichelmann J, Berndt S, Pott C, Eisenbarth T (2021) Help, my Signal has bad Device! Breaking the Signal Messenger's Post-Compromise Security Through a Malicious device. In Detection of Intrusions and Malware, and Vulnerability Assessment: 18th International Conference, DIMVA 2021, Virtual Event, July 14–16, 2021, Proceedings, Springer-Verlag, Berlin, Heidelberg, 88–105
32. Vazquez Sandoval I, Atashpendar A, Lenzini G, Ryan PYA (2021) PakeMail: authentication and key management in decentralized secure email and messaging via PAKE. In: Obaidat MS, Ben-Othman J (eds) E-Business and telecommunications. Springer International Publishing, Cham, pp 102–128
33. Müller M, Lee JU, Frick N, Stangier L, Gurevych I, Metternich J (2021) Extracting problem related entities from production chats to enhance the data base for assistance functions on the shop floor. Procedia CIRP 103:231–236. https://doi.org/10.1016/j.procir.2021.10.037
34. Eisele S, Eghtesad T, Campanelli K, Agrawal P, Laszka A, Dubey A (2021 December) Safe and private forward-trading platform for transactive microgrids. ACM Trans Cyber-Phys. Syst 5(1). https://doi.org/10.1145/3403711
35. Brodbeck F, Hromada DD (2022) February. teacher.Js: a low-bandwidth digital Tool for outdoor online teaching. In 2022 IEEE 5th Eurasian Conference on Educational Innovation (ECEI), pp 47–50

36. Ihle C, Deifuß F, Schubotz M, Gipp B (2022 July) Towards portable identities in the Matrix protocol. In 2022 IEEE 42nd International Conference on Distributed Computing Systems Workshops (ICDCSW), pp 88–89. ISSN:2332-5666
37. Griggio CF, Nouwens M, Klokmose CN (2022) Caught in the network: the impact of WhatsApp's 2021 privacy policy update on users' messaging app ecosystems. In Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems, CHI'22, Association for Computing Machinery. event-place, New York, NY, USA.: New Orleans, USA
38. Chowdhury PD, Sameen M, Blessing J, Boucher N, Gardiner J, Burrows T, Anderson R, Rashid A (2023 May) Threat models over space and time: a case study of E2EE messaging applications. arXiv:2301.05653 [cs
39. Alwen J, Mularczyk M, Tselekounis Y (2023) Fork-resilient continuous group key Agreement. In Advances in Cryptology – CRYPTO 2023: 43rd Annual International Cryptology Conference, CRYPTO 2023, Springer-Verlag. event-place, Santa Barbara, CA, USA,, Berlin, Heidelberg, August 20–24, 2023, Proceedings, Part IV, 396–429
40. Gibb R, Madhavapeddy A, Crowcroft J (2023) Where on Earth is the Spatia name system? In Proceedings of the 22nd ACM Workshop on Hot Topics in Networks, HotNets'23, New York, NY, USA,: Cambridge, MA, USA., 79–86. Association for Computing Machinery. event-place
41. Wallez T, Protzenko J, Bhargavan K (2023) Comparse: provably secure formats for cryptographic protocols. In Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS'23, New York, NY, USA,: Copenhagen, Denmark., 564–578. Association for Computing Machinery. event-place
42. Cui M, Gu C, Nan G (2023 March) CrossChat: instant messaging across different apps on mobile devices |. In Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing
43. Rahman M, Wang Y, De D (2024) Implementation of dew-inspired matrix-mesh communication protocol. In: Roy S (ed) Dew computing: the sustainable IoT perspectives. Springer Nature Singapore, Singapore, pp 105–115. https://doi.org/10.1007/978-981-99-4590-0_5
44. Blessing J, Chowdhury PD, Sameen M, Anderson R, Gardiner J, Rashid A, Stajano F (2023) Towards Human-centric endpoint security. In: Matyáš V, Christianson B, Anderson J (eds) Security protocols XXVIII, vol 14186. Springer Nature Switzerland. Series Title: Lecture Notes in Computer Science. Cham, pp 211–219. https://doi.org/10.1007/978-3-031-43033-6_21
45. Albrecht MR, Dowling B, Jones D (2024 May) Device-oriented group messaging: a formal cryptographic analysis of matrix core. In 2024 IEEE Symposium on Security and Privacy (SP), IEEE
46. Franco M, Gaggi O, Guidi B, Michienzi A, Palazzi CE (2023 August) A decentralised messaging system robust against the unauthorised forwarding of private content. Future Gener Comput Syst 145:211–222. https://doi.org/10.1016/j.future.2023.03.025

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

A Comparative Performance Study of the Matrix `/sync` Endpoint on Synapse and Tuwunel

Windson Viana[1], Francisco Airton Silva[2], Francisco A. A. Gomes[1], Michel Sales[1], Roberto Ivo[1], Fernando A. M. Trinta[1], Rossana M. de C. Andrade[1], José A. F. de Macêdo[1], Vinícius Lagrota[3], Rodrigo Pacheco[3] and Paulo A. L. Rego[1],

[1]Universidade Federal do Ceará (UFC)

[2]Universidade Federal do Piauí (UFPI)

[3]Centro de Pesquisa e Desenvolvimento para a Segurança das Comunicações (CEPESC)

{windson,michelsb}@ufc.br

{fernando.trinta,jose.macedo,rossana, paulo}@dc.ufc.br

robertoivo@insightlab.ufc.br, almada@crateus.ufc.br

faps@ufpi.edu.br {vinicius.1232,rodrigo.6874}@abin.gov.br

Abstract. *Matrix is an open protocol for federated communication, primarily implemented by the Synapse server, with Tuwunel as an alternative. Client–server interaction relies on the synchronization (sync) mechanism, which delivers events and updates throughout the application lifecycle. This paper presents a controlled experimental comparison of Synapse and Tuwunel based on nearly 17,000 sync requests under increasing load and different filtering strategies. Results show that Tuwunel achieves lower response times in three of four configurations, but it does not yet support Simplified Sliding Sync (SSS) correctly. SSS on Synapse delivers the lowest latency across the evaluated settings. We also report saturation and resource trends under increasing load.*

1. Introduction

Matrix[1] is an open protocol for federated communication, widely adopted in distributed messaging systems and increasingly leveraged by governmental initiatives seeking communication channels with greater autonomy, interoperability, and control over data and infrastructure [Martins et al. 2026]. It's the foundation for several large-scale applications, including Tchap[2], used by the French government, with over 600,000 users, the BwMessenger[3] in Germany, and msg gov[4], an official messaging solution developed by the Brazilian government, demonstrating its suitability for sovereign and secure communication environments. Matrix also underpins commercial clients such as Element and Element X[5], which together support millions of users worldwide [Martins et al. 2025].

Synapse[6] is its most widely used server implementation. In parallel, Tuwunel[7] has

[1]https://matrix.org/
[2]https://tchap.numerique.gouv.fr/
[3]https://tinyurl.com/bwmessenger
[4]https://tinyurl.com/newsmsggov
[5]https://element.io/
[6]https://github.com/matrix-org/synapse
[7]https://github.com/matrix-construct/tuwunel

emerged, proposing distinct architecture with the goal of simplifying operation and reducing execution costs. Although both implement the same protocol, differences in design, component organization, and processing strategies can lead to distinct behaviors under load not yet well documented. At the center of the interaction between Matrix clients and servers is the `/sync` endpoint, responsible for delivering events, state, and updates needed to maintain a consistent view of rooms. Due to its reliance on polling mechanisms, this mechanism (*sync*) is the most frequently accessed endpoint on servers, in some cases accounting for more than 75% of all requests. This mechanism directly affects the volume of data transferred, the response time perceived by the client, and the consumption of resources on the server. Several synchronization strategies have been proposed in Matrix, including the use of filtering, deferred data delivery, and, more recently, pagination of the number of synchronized rooms, i.e., the Simplified Sliding Sync (SSS)[8].

Despite the relevance of *sync* for the user experience and for the operational cost of a Matrix server, there is still a lack of experimental characterization that directly connects the behavior observed on the client side (response time and volume of returned data) to the load imposed on the server (CPU and memory usage), especially when considering the adoption of more recent mechanisms such as SSS. Furthermore, it is not clear to what extent Tuwunel differences are reflected in saturation patterns, payload growth, and resource consumption as the amount of data to be synchronized increases.

This work presents a controlled experimental evaluation of the Matrix synchronization mechanism, comparing Synapse and Tuwunel under equivalent conditions and with a progressively increasing load of data to be synchronized. The proposal is to analyze how synchronization behaves as the volume of information returned to the client grows and as the computational effort required from the server increases, highlighting behavioral differences between the implementations. In this way, the article provides quantitative evidence that supports both the choice of server and the understanding of the cost of *sync* in intensive-use scenarios. This study is part of the msg gov project [9] and reflects real operational requirements rather than a purely external benchmark.

The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 details the *sync* in Matrix, establishing the concepts needed to interpret the results. Section 4 describes the architecture explored and the components relevant for comparing servers. Sections 5 and 6 present the experimentation methodology and its results. Finally, Section 7 concludes the paper with the main findings and future directions.

2. Related Work

Several studies were identified as aligned with this paper's goals [Martins et al. 2026]. Three of them analyze the distribution of users across rooms and servers in the Matrix ecosystem or fault tolerance [Jacob et al. 2019, Jacob et al. 2021, Stocker et al. 2025], contributing to an understanding of how the platform behaves as the number of users, rooms, and events increases. Jacob et al. [Jacob et al. 2019] studied the scalability of Matrix as a message-oriented synchronization middleware in federated platforms, discussing how synchronization and event volume can put pressure on the server and affect response times in scenarios close to real-world usage. In [Jacob et al. 2021], the authors investigate the Matrix Event Graph (MEG) as a replicated data type for decentralized apps, validating

[8]https://github.com/matrix-org/sliding-sync

[9]https://tinyurl.com/msggovapp

it as a Conflict-free Replicated Data Type (CRDT) with strong eventual consistency. The work helps contextualize costs associated with replication and concurrency.

[Stocker et al. 2025] investigated Synapse's behavior under crash-type failures, in which the process stops responding. The proposal evaluated to what extent a user's server downtime affects their communication and whether the protocol is able to recover undelivered messages after the server comes back online. Three other works emphasize security and privacy in Matrix [Müller et al. 2021, Albrecht et al. 2023]. Muller et al. [Müller et al. 2021] dealt with the incorporation of informal communications into digital industrial management systems; although it is not a Matrix-centered study, it reinforces the presence of messaging platforms in operational workflows and the need to treat them as part of the support infrastructure. Albrecht et al. [Albrecht et al. 2023] showed vulnerabilities in the Matrix protocol and in the Element client, highlighting the effects of malicious servers on confidentiality and authentication guarantees; this type of analysis is relevant because security requirements and implementation choices can affect critical routes and, consequently, the performance perceived by the user.

Unlike previous studies, which tend to emphasize dimensions such as the structural scalability of the Matrix ecosystem or security and privacy concerns, this work focuses on a gap of an operational nature: how the synchronization mechanism behaves as data volume and concurrency increase. The focus is on the `/sync` endpoint and its more recent variant, SSS, as they are central components of client–server interaction and directly influence perceived latency, the size of the returned payload, and the computational cost on the server. In this sense, the comparison between Synapse and Tuwunel is treated as part of the problem, since implementation differences can produce distinct responses under the same load conditions.

3. *Sync* Matrix Function

In this work, we focus on the synchronization feature: the **sync** endpoint [10]. It is the main way in which Matrix clients receive from the homeserver the events necessary to keep the local state updated, including new messages, room state changes, and other notifications associated with the user session. Figure 1 illustrates the basic flow of this behavior: Client A sends a message ("Hello B, how are you?") to a room shared with Client B, and the homeserver registers this event. Then, Client B does not "receive" the message via a direct push from the server; instead, it executes a pooling *sync* request to the homeserver, and the *sync* response contains the newly created event, allowing the client to update its timeline and/or notifications.

The *sync* feature acts as the event delivery mechanism for the client, and the operating cost increases as active users, rooms, and accumulated events between synchronizations grow. Because *sync* is triggered both periodically (e.g., every 30 seconds) and in response to specific events, such as after login, when entering rooms, and during user interactions, it tends to concentrate a large portion of the client–server traffic; becoming crucial for perceived latency and overall server load. From an implementation standpoint, *sync* on homeserver assembles a JSON response aggregating events from multiple rooms linked to the user. The volume and cost of this response vary with factors such as: number of rooms, number of pending events per room, limits on returned messages, applied filters, and inclusion of additional data (e.g., member list). For example, if between the

[10] `https://spec.matrix.org/v1.17/client-server-api/#syncing`

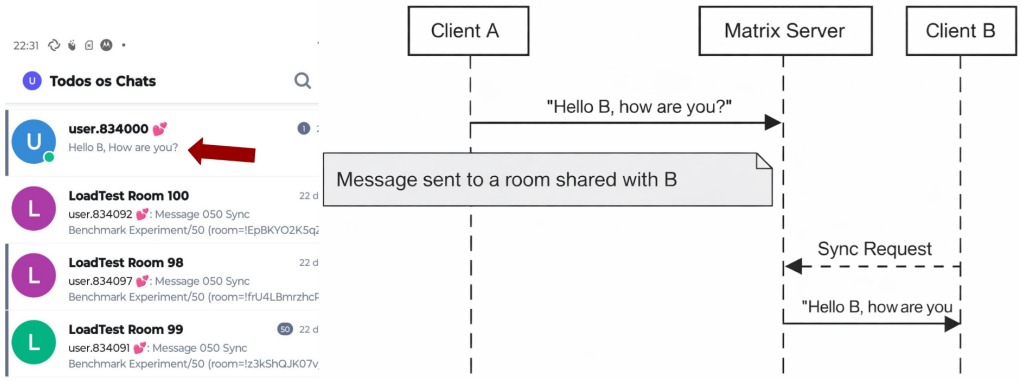


Figure 1. Basic functionality of *sync*

last and the current *sync*, the user received N messages across M rooms, the server will decide, depending on the *sync* strategy, which of these N messages will be sent. For instance, the server may choose to send only the most recent message received from each of the rooms the user is a member of. Under high load, this process puts pressure on CPU, memory, and database, as each *sync* involves queries and the assembly of large structures, and differences in sync strategy can directly alter the response time and server stability.

3.1. Filters Timeline Limit and Lazy-Loading Room Members

In the Matrix protocol, the `/sync` endpoint provides a filtering mechanism that allows clients to precisely control the information returned during synchronization, limiting transferred data volume, reducing JSON response size, and adjusting server-side processing costs. For instance, the `timeline.limit` filter defines the maximum number of events returned per room, where smaller values reduce payload size and serialization overhead at the cost of requiring additional room-specific requests, while larger values increase transferred history and server load.

The `state_lazy_load_members` filter defines whether the state information related to room members should be loaded lazily or completely. When set to **true**, the server avoids including all member state events in the initial response, returning only those strictly necessary, which reduces the volume of data in rooms with many participants. When set to **false**, the complete member state is sent, increasing processing and transfer costs, but providing a more comprehensive view of the room state in each sync.

The chosen synchronization strategy directly affects the number of missing messages/events that must be fetched when a user enters a specific conversation. If the synchronization process returns only a single message per room, entering a room requires loading a larger number of additional messages and events (e.g., a User C is a new member of the room). For example, if only one message is retrieved during synchronization, the client must request the remaining 49 messages upon room entry. In contrast, strategies that return a larger portion of each room's timeline, such as those used by clients like Element Web, may retrieve 20 messages per room during synchronization. In this case, entering a specific room requires fetching only the remaining 30 messages, thereby reducing both latency and server workload at the moment of room access.

The combination of `timeline.limit` and `state_lazy_load_members` allows exploring different synchronization strategies, ranging from minimal and highly

restricted responses to more complete and costly synchronizations. These selected combinations emulate configuration choices available in clients such as Element Classic, Element Web, and Fluffy. By testing these combinations on both Synapse and Tuwunel servers, it becomes possible to compare how each implementation handles the progressive increase in data load, identifying differences in scalability, response time, and how filters influence the cost of syncing in intensive usage scenarios.

3.2. Simplified Sliding Sync - SSS

SSS is an evolution of the traditional Matrix sync mechanism that seeks to more precisely control the volume of data exchanged between client and server as the number of rooms grows. Instead of uniformly synchronizing all rooms, SSS introduces the concept of a sliding window, in which the server defines which subsets of rooms should be synchronized in each request. This approach follows the same principle as the `/sync` filters, that is, limiting transferred data and processing cost, but applied directly to room selection. Instead of synchronizing all rooms at once, the client initially requests a subset of the room list, typically limited to the first 20 rooms, with at most **one message per room** included in the response. As interaction progresses, additional synchronization windows are requested, gradually expanding the synchronized set to larger ranges, commonly increasing to 100 and then 200 rooms, until full coverage of the room list is achieved.

From a strategy perspective, SSS shifts synchronization from a bulk, room-complete model to an incremental, window-based approach. This design reduces the size of individual responses and distributes both network and server-side processing costs over time. A drawback of this strategy is that it increases the number of synchronization requests and requires server-side ordering logic to determine which rooms should be returned in each window. SSS is specified in the Matrix Client–Server API[11] and, at the time of writing, remains a beta feature, with ongoing refinement and partial adoption across Matrix clients. Another drawback of SSS is that it is possible for a user to enter a room for which no messages have been retrieved during synchronization. In such scenarios, when the user clicks to open the room, the client triggers the retrieval of all missing messages and events.

4. Infrastructure and Implementation of the Assessment Environment

Figure 2 shows the test scenario. The experiments were conducted in a local network environment in which one machine was dedicated to simulating clients issuing synchronization and room access requests, while a separate machine was used to host the Synapse and Tuwunel servers. Synapse is the reference Matrix homeserver implementation, written in Python, and is widely adopted in production deployments. In our setup, Synapse was backed by a PostgreSQL database, which is the recommended configuration for production environments and supports high concurrency and persistence requirements. Tuwunel is an alternative Matrix server implementation written in Rust, designed with a focus on performance and resource efficiency. In our experiments, Tuwunel used RocksDB as its database backend, following the default installation configuration. This database, unlike PostgreSQL, is a key-value store focused on high read performance. However, it has limitations in terms of query capabilities and index usage. Tuwunel is tightly coupled to this implementation. In this sense, our experiments are comparing not only the servers, but

[11] https://tinyurl.com/sssmatrix

rather the server–storage pairs (Synapse, Tuwunel) and their standard storage solutions (PostgreSQL, RocksDB).

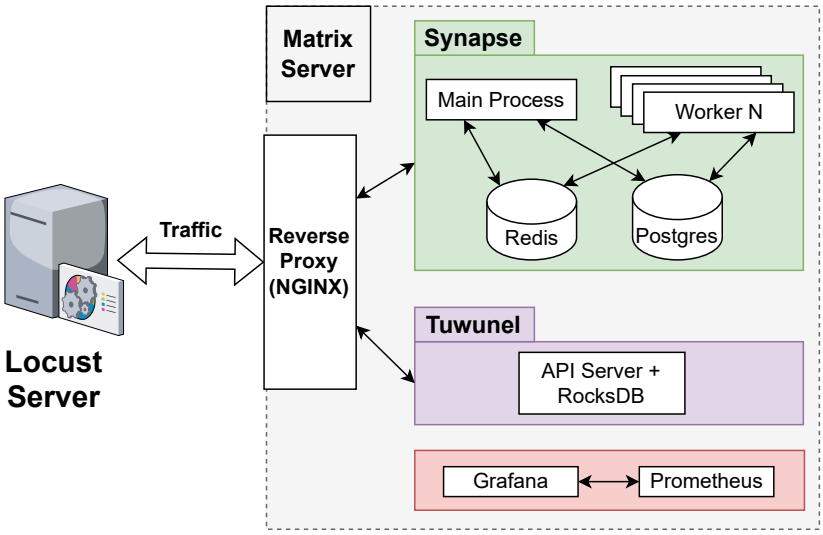


Figure 2. Matrix architecture used in the test environment.

The server machine used an Intel(R) Core(TM) i9-14900K processor with 24 cores, 32 threads, and 192 GB of main memory. The experimental setup deployed both Tuwunel and Synapse on this machine and ran Ubuntu Server 24.04 LTS as the operating system. This hardware and software configuration provided substantial computational capacity and memory resources. The setup executed both servers inside virtualized container environments with different configuration setups. Based on prior studies, the setup adopted a high-scalability architecture for Synapse, composed of the Main Process and multiple workers. In contrast, we adopted a monolithic configuration for Tuwunel, following the default installation instructions. In addition, each architectural component illustrated as a white box in Figure 2 ran as an individual container.

In our experiment, Synapse was deployed using a modular and scalable configuration inspired by real production environments. At its core, the Main Process acts as a coordination unit, responsible for routing client requests, managing federated communication, and orchestrating interactions among worker services. It maintains continuous communication with the data layer and distributed services, enabling efficient task delegation under high demand. To maximize CPU utilization, this experimental setup distributes responsibilities across 18 specialized workers. Examples include background workers for asynchronous tasks, media repository workers for handling media content, federation sender workers for inter server communication, and generic workers that support client requests and event processing (e.g., event creation and client reading).

To enable observation of system behavior during experiments, Prometheus collects real-time metrics from Synapse and Tuwunel services and the container engine itself. Metrics such as CPU and memory usage, request throughput, and response latency are collected in detail, allowing a comprehensive view of system behavior. This information is presented through Grafana, which offers dynamic dashboards for monitoring system health, identifying bottlenecks, and supporting performance analysis.

5. Experiment Methodology

5.1. Research Questions

RQ1: What is the impact of the `timeline.limit` parameter and the `state_lazy_load_members` filter on the performance of the initial sync? This research question investigates how different synchronization configurations affect both client-side latency and server-side resource consumption.

RQ2: What are the performance differences between Synapse and Tuwunel for synchronization operations?

The Matrix protocol is implemented by multiple server implementations with distinct architectural choices that may influence overall performance.

RQ3: What are the performance differences between classic synchronization strategies and Simplified Sliding Sync?

Simplified Sliding Sync aims to reduce the size of synchronization responses by limiting the number of rooms and messages returned per request; however, its impact on server-side performance and the extent to which it improves client-side response times are not yet well understood.

5.2. Rooms and Number of Users

Two user groups were created to populate the Synapse and Tuwunel databases. The **first group** was composed of 100 users, each belonging to 100 rooms, where each room contains all 100 members. The **second group** was composed of 300 users, each belonging to 300 rooms, where each room contains 300 members. In both groups, each room was populated with 50 messages. The user responsible for writing the 50 messages was randomly selected during scenario generation. 50 was chosen because it corresponds to the default pagination behavior of the Element Classic client, which loads the most recent 50 messages when entering a room. Messages follow the template:

```
Message ID Sync Benchmark Experiment/TOTAL
```

This fixed template ensures message content remains consistent across runs, so differences in payload size are driven primarily by synchronization strategy and state inclusion rather than by message variability. The **first group** of users was designed to simulate heavy users who participate in multiple work-related groups, where rooms typically have a moderate number of members, but the user is involved in many parallel conversations. The **second group** aims to represent heavy users engaged in large groups with a high number of participants, reflecting environments with high fan-out and increased synchronization complexity. It is worth noting that real-world deployments can exhibit even larger groups; for instance, the official Matrix room on Element has more than 5,000 members.

5.3. User Simulation

A Python-based Locust workload [12] was used to simulate users accessing the server in parallel while employing the same synchronization strategy under evaluation. The following scenarios were defined:

[12]Locust is an open-source, Python-based load testing framework that enables the simulation of concurrent users and the definition of user behavior through programmable scenarios, making it suitable for evaluating the performance and scalability of web services.

- **C1:** A single user performs an initial synchronization and requests the messages of a room. The user can be of both groups.
- **C100:** 100 users of the first group perform an initial synchronization and request the messages of a room.
- **C200:** 200 users of the second group perform an initial synchronization and request the messages of a room.

In all three cases, when entering a specific room, clients request the set of messages necessary to complete the full message history. In this sense, strategies that prioritize sending fewer messages during the initial sync will need to request more messages when entering a specific room.

5.4. Sync strategies

Beyond the use of SSS, the experiments explore four distinct configurations of *sync*, obtained from the combination of two filters used by the protocol: `timeline.limit`, which controls the number of events returned per room, and `state_lazy_load_members`, which defines the state loading strategy for members. These configurations are compactly identified throughout the graphs by labels such as 1T, 5T, 10F, and 20T (see Table 1 for more details), allowing comparison of scenarios with different timeline depths and state loading strategies. 10F corresponds to the Element Classic configuration by default, and 20T the Element Web configuration.

Table 1. Filter settings used in the *sync* experiments.

Config.	timeline.limit	lazy load members	Description
1T	1	true	Returns only the most recent event per room, using lazy loading for member state.
5T	5	true	Returns up to five events per room, while maintaining lazy loading of member state.
10F	10	false	Returns up to ten events per room, including the full member state in the synchronization.
20T	20	true	Returns up to twenty events per room, using lazy loading for member state.

5.5. Preliminary tests

For each scenario and server, preliminary tests were conducted to verify correctness and execution conditions. First, these tests revealed that a scenario with 300 users of the second group (with 300 rooms) generated a large number of failures on Synapse, which led us to evaluate only 200 users of this group. Additionally, SSS did not complete on Tuwunel in the 300-room dataset. We reproduced this behavior using Element X connected to Tuwunel, where synchronization consistently stalled after the first window, effectively limiting correct synchronization to the initial 20 rooms. In the C200 scenario, SSS failed to complete on Synapse due to limitations in the default SSS configuration, with most requests returning 504 Gateway Time out. After enabling a worker based configuration for SSS, synchronization completed correctly.

5.6. Procedure

We started Locust with a randomly selected combination of scenario and sync strategy (e.g., 1T with 200 users from the second group) and executed separately for each server. A report was generated for each execution, containing server response times, response JSON sizes, and CPU and memory utilization metrics. Locust was then restarted for the

next scenario and synchronization strategy. Each configuration was executed five times, resulting in a total of 110 executions. 6,000 synchronization requests were issued for each server, covering the four filter combinations, in addition to 5,000 SSS requests executed exclusively on Synapse. Execution times ranged from 12 seconds for scenario C100 with the 1T strategy to approximately 13 minutes for the 10F strategy on Tuwunel. For scenario C1, due to short response times, each configuration was executed 10 times on each server.

6. Results

6.1. RQ1: What is the impact of the `timeline.limit` parameter and the `state_lazy_load_members` filter on the performance of the initial sync?

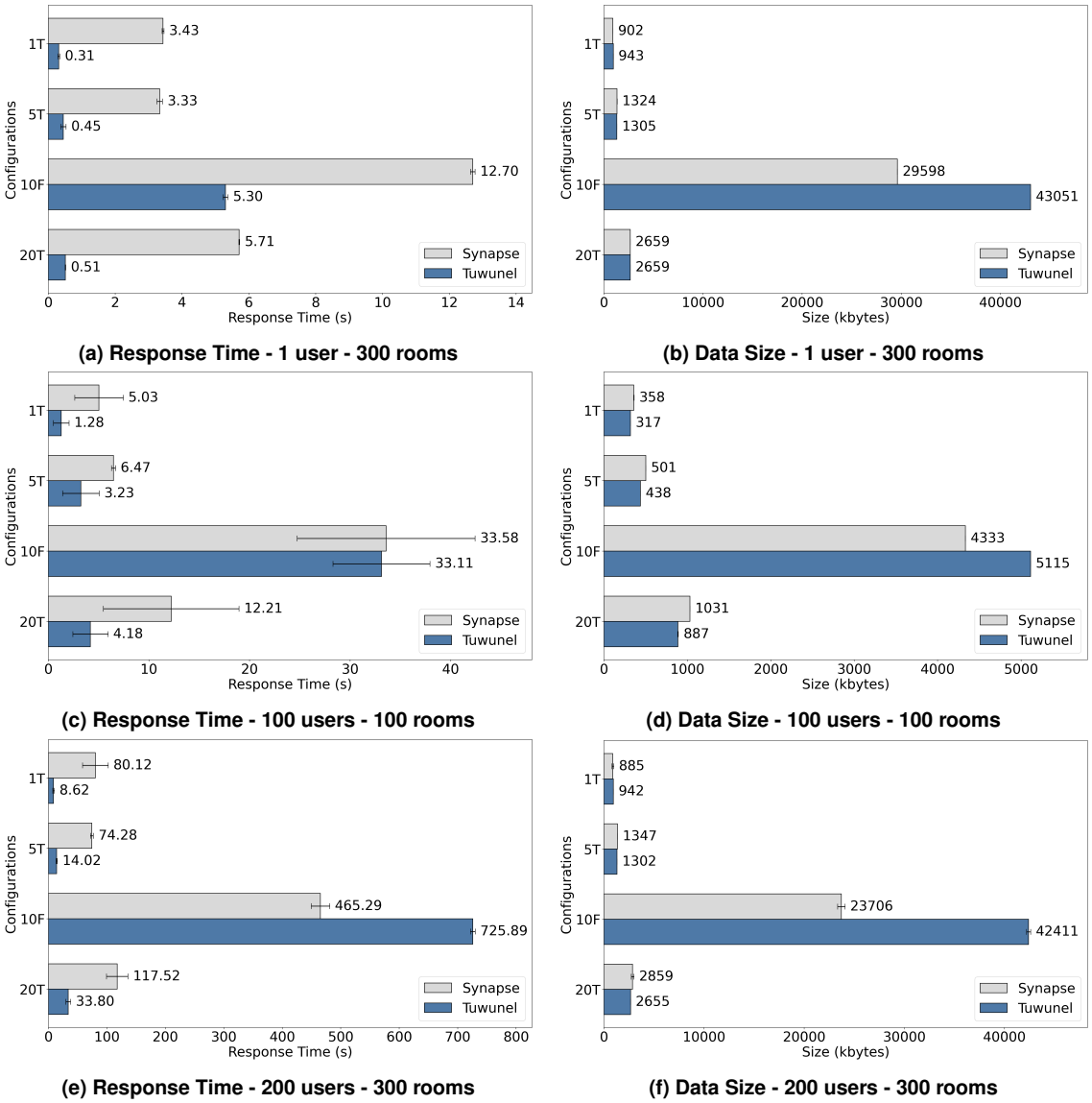
Main result: The use of the `state_lazy_load_members` filter has the most impact on the sync strategies, with configuration 10F exhibiting the poorest performance. By contrast, limiting the number of messages has a comparatively smaller impact; however, it still yields measurable effects on server response time.

The six graphs in Figure 3 organize these results into three load scenarios, simultaneously varying the number of users and the number of rooms associated with each execution. In particular, the first two graphs consider the scenario with 1 user and 300 rooms (C1), the two central graphs represent the scenario with 100 users and 100 rooms (C100), and the last two correspond to the scenario with 200 users and 300 rooms (C200).

Figures 3a and 3b present the scenario with 1 user and 300 rooms, allowing for isolated observation of the impact of the synchronized context volume on *sync*, without the interference of concurrency between multiple clients. Graph 3a shows the response time, in which Synapse shows a noticeable increase as the `timeline.limit` grows, with the 10F configuration standing out, concentrating the longest response time among the four configurations evaluated. **Tuwunel maintains significantly shorter times** in all configurations, although it also shows variation according to the timeline depth. Figure 3b, which shows the amount of synchronized data, shows that the size of the responses increases sharply when the limit of events per room increases, again with the 10F configuration standing out as the most costly in terms of data volume. In this case, Tuwunel returns larger responses than Synapse, indicating differences in how each implementation serializes and organizes the sync data when the lazy loading members filter is not used.

In Figure 3c and 3d, the scenario with 100 users and 100 rooms is analyzed, in which the *sync* simultaneously reflects the effect of concurrency between clients and the volume of data associated with each room. In Figure 3c, which shows the response time, it can be observed that all configurations show an increase compared to the scenario with only one user, with emphasis on configuration 10F, which concentrates the longest times in both Synapse and Tuwunel. In Figure 3d, referring to the amount of synchronized data, the behavior varies more subtly between the implementations. Configuration 10F again stands out as the most costly in terms of volume, with higher response rates in Tuwunel. In contrast, in configuration 20T, Synapse returns a higher volume of data than Tuwunel, indicating that, even with lazy loading enabled, increasing the event limit per room is sufficient to reverse the pattern observed in other configurations.

In Figures 3e and 3f, the most heavily loaded scenario is considered, with 200 users and 300 rooms, in which the *sync* starts operating under high concurrency and a large volume of synchronized context. In Figure 3e, which shows the response time,



(a) Response Time - 1 user - 300 rooms

(b) Data Size - 1 user - 300 rooms

(c) Response Time - 100 users - 100 rooms

(d) Data Size - 100 users - 100 rooms

(e) Response Time - 200 users - 300 rooms

(f) Data Size - 200 users - 300 rooms

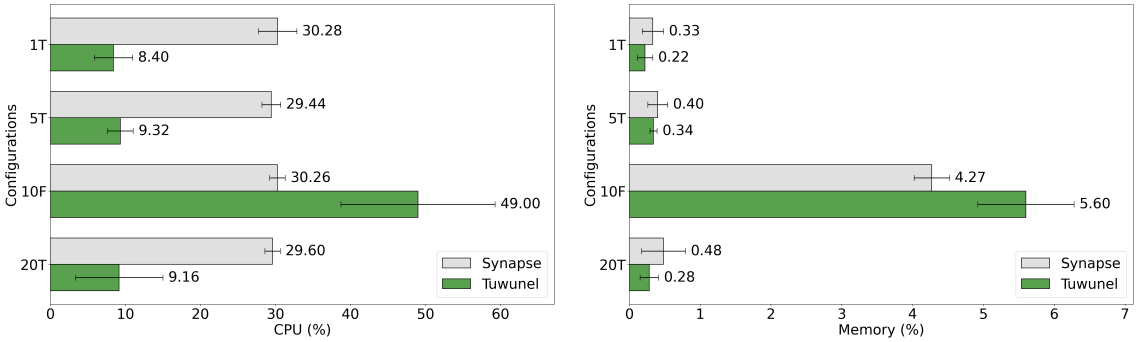
Figure 3. Synapse versus Tuwunel focusing on response time and data size

a significant increase is observed in all configurations, especially in configuration 10F, which concentrates the highest values. It is worth noting that, in all executions of the 10F strategy on Synapse, approximately 1% of the requests resulted in failures due to timeouts. Timeouts were recorded as failures by Locust and excluded from average response-time calculations. Unlike the scenario with 100 users, Synapse shows a lower response time than Tuwunel in 10F, indicating a reversal in the relative behavior of the implementations under higher load pressure. Configurations 1T and 5T also show a substantial increase in latency compared to the previous scenarios, but maintain the same order of magnitude between servers, while 20T appears again at an intermediate level.

In Figure 3f, referring to the amount of synchronized data, the behavior between Synapse and Tuwunel becomes closer in several configurations. The 10F configuration remains the most costly in terms of volume, with Tuwunel returning larger responses, reinforcing the impact of full member state loading in high concurrency scenarios.

6.2. RQ2: What are the performance differences between Synapse and Tuwunel for synchronization operations?

Main result: Tuwunel exhibited lower response times than Synapse in scenarios that employed the `state_lazy_load_members` filter. While both servers showed minimal variation in memory consumption, an increase in CPU usage was observed. In the C200 scenario, Synapse consumed approximately three times more CPU resources than Tuwunel, with the exception of configuration 10F.



(a) CPU - 200 users - 300 rooms **(b) Memory - 200 users - 300 rooms**

Figure 4. CPU and Memory variation

Figure 4a reports the average CPU consumption variation for Synapse and Tuwunel. Overall, CPU increase remains consistently high for both servers, reinforcing that *sync* processing represents a substantial workload on the backend. The 10F configuration, which concentrates a larger volume of synchronized data in each request, triggers a pronounced rise in CPU usage on Tuwunel, suggesting a stronger sensitivity of this implementation to configuration changes and heavier synchronization workloads. In contrast, Synapse exhibits relatively stable CPU increase across the evaluated configurations, indicating that, within the tested range, increasing the synchronized-data volume does not translate into a proportional increase in CPU demand on this server.

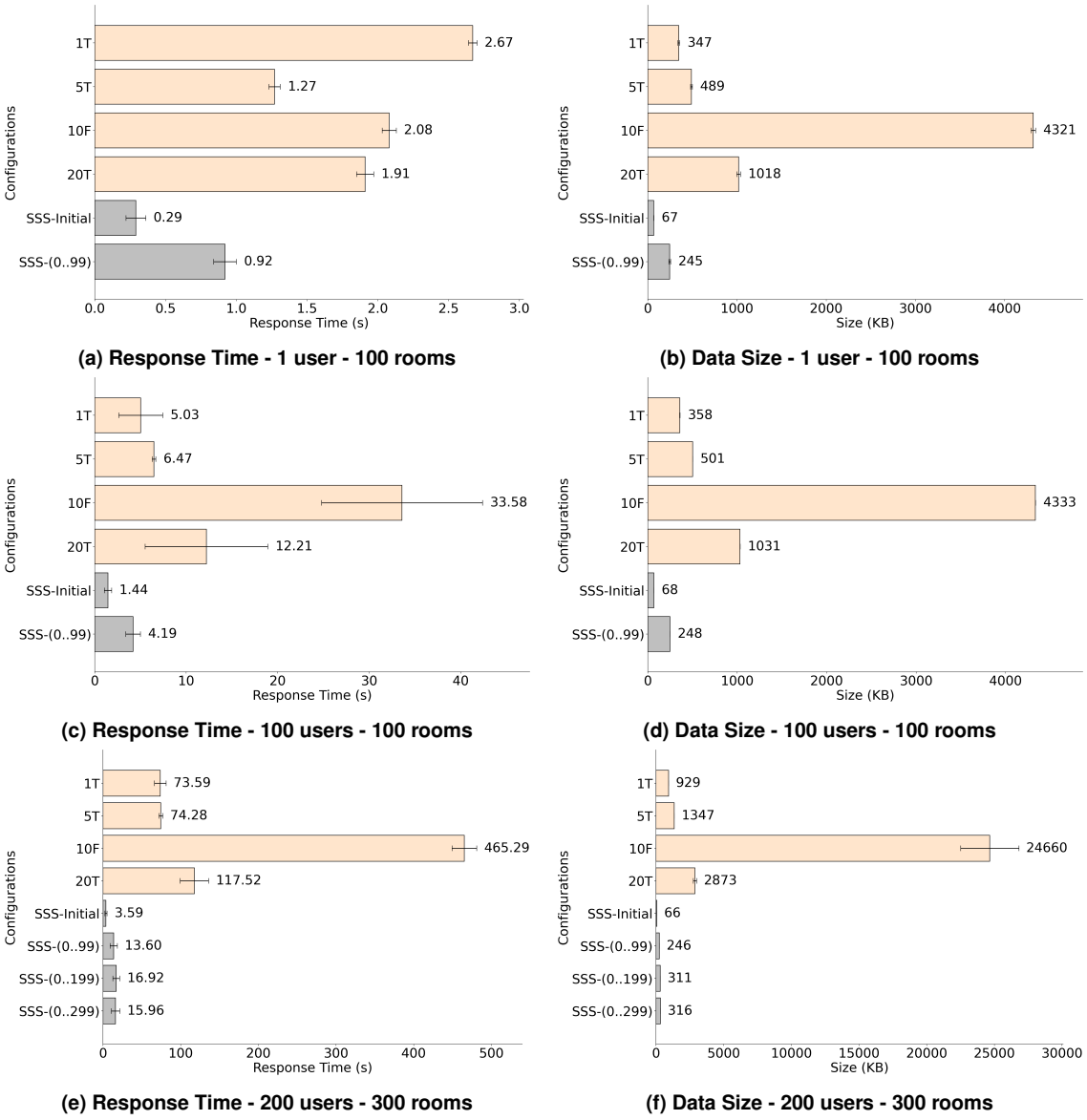
Figure 3b presents the average increase of memory consumption during the same experiments. Compared to CPU, memory usage stays low across all scenarios, pointing to limited memory pressure even when synchronization traffic grows. Still, a clearer increase appears again in the 10F configuration for both servers, consistent with the larger amount of state and events processed per *sync* response. Even in this heavier configuration, memory consumption increases modestly around 4% for Synapse and 5% for Tuwunel, indicating that the *sync* mechanism is predominantly CPU-bound in these settings, while its impact on memory remains comparatively small.

6.3. RQ3: What are the performance differences between classic synchronization strategies and Simplified Sliding Sync?

The results indicate that SSS generally reduces the amount of data transferred and delivers the shorter response times under higher concurrency.

Figure 5 presents a performance analysis of Synapse when using the traditional *sync* endpoint and the SSS one, considering both response time and synchronized data size under different workloads. By pairing latency and payload volume, the figure enables a

direct interpretation of how synchronization strategies impact user-perceived performance and server-side costs as the number of rooms and users increases.



(a) Response Time - 1 user - 100 rooms **(b) Data Size - 1 user - 100 rooms**

(c) Response Time - 100 users - 100 rooms **(d) Data Size - 100 users - 100 rooms**

(e) Response Time - 200 users - 300 rooms **(f) Data Size - 200 users - 300 rooms**

Figure 5. SSS detailed performance evaluation.

Figures 5(a)–(f) show that, across all evaluated scenarios, each Simplified Sliding Sync (SSS) request exhibits substantially lower response times when compared to traditional *sync* configurations. This behavior is consistent even under higher concurrency levels, as illustrated in Figure 5(c), where traditional *sync* configurations with lazy-loaded members achieve lower response times only when considering a single, monolithic request. In contrast, SSS decomposes synchronization into multiple lightweight requests, effectively distributing and spacing the load on the client side. While this approach may increase the total time required for all messages to be delivered, it significantly improves responsiveness at each synchronization step, reducing perceived latency and enabling smoother client interaction under load.

In our server configuration, the adoption of SSS does not introduce a substan-

tial increase in resource consumption when compared to other synchronization strategies. CPU usage followed trends similar to the traditional configurations, with an observed increase of approximately 30% under higher concurrency, while memory consumption remained largely stable, varying by around 0.3%. These results indicate that, despite the higher number of synchronization requests issued by SSS, its incremental and windowed design avoids additional pressure on server resources, maintaining a resource profile comparable to that of conventional *sync* approaches. It is worth noting that if we sum all SSS requests, we approach the 1T option, with the advantage that the UI can already display part of the data to the user. The downside is if the rooms of interest to the user only appear in the final SSS requests, which would force the user either to wait or to request the room's messages upon clicking, thereby breaking the initial sync logic.

7. Final Considerations

This work presented an experimental evaluation of the *sync* mechanism in the Matrix protocol, comparing Synapse and Tuwunel under varying workloads (number of users and rooms) and synchronization strategies. The analysis encompassed response time, synchronized data volume, server-side CPU and memory consumption, as well as a characterization of the SSS mechanism.

The results show that the cost of *sync* increases with higher data volumes and concurrency levels. The 10F configuration yields the highest response times and response sizes. Overall, Tuwunel achieved lower response times in less loaded scenarios and exhibited lower variation in CPU usage across configurations, although it showed increased sensitivity as the load intensified. Synapse, in contrast, maintained consistently stable CPU consumption across configurations. For both servers, memory consumption exhibited limited variation, even under the most demanding scenarios.

The SSS evaluation revealed an incremental, window-based behavior that distributes synchronization costs over time by decomposing sync into lightweight requests. While this approach may increase the overall time required for all messages to be delivered under high concurrency, each individual request exhibits significantly lower response times. Despite the higher number of sync requests, no additional resource consumption was observed, highlighting trade-offs between per-request responsiveness and total synchronization completion time for homeserver selection and sync configuration in intensive usage scenarios. It is important to note, however, that these results were only achievable due to the adoption of a specific distribution configuration, as current documentation does not clearly describe how this configuration should be optimally applied. Once a correct and well-documented implementation becomes available in Tuwunel, the expectation is that its performance will further improve, potentially surpassing Synapse by following the performance trends observed in other synchronization strategies.

Some limitations of this study should be acknowledged. We simulated heavy users under specific configurations, which may not fully represent the diversity of usage profiles observed in messaging applications deployed in governmental services. We compared the configurations recommended by the Synapse and Tuwunel teams. There is an expectation that Tuwunel's performance can be further improved by increasing the number of workers used. However, the lack of indexing in the database may become detrimental as the number of users and the volume of data on the servers grow significantly. Measuring variations in memory and CPU consumption is inherently challenging, as other system services or operating system functionalities may execute concurrently and interfere with resource us-

age. Despite these limitations, repeated experimental runs consistently exhibited the same behavioral patterns across synchronization strategies and server implementations, thereby strengthening the support for the conclusions presented.

As future work, we plan to investigate the impact of synchronization algorithms on mobile and web clients in terms of processing time and resource consumption, as well as to propose strategies to reduce the size of the JSON-based payload.

References

Albrecht, M. R., Celi, S., Dowling, B., and Jones, D. (2023). Practically-exploitable cryptographic vulnerabilities in matrix. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 164–181. IEEE.

Jacob, F., Beer, C., Henze, N., and Hartenstein, H. (2021). Analysis of the matrix event graph replicated data type. *IEEE access*, 9:28317–28333.

Jacob, F., Grashöfer, J., and Hartenstein, H. (2019). A glimpse of the matrix (extended version): Scalability issues of a new message-oriented data synchronization middleware. *arXiv preprint arXiv:1910.06295*.

Martins, J. A., Rego, P. A., de Macêdo, J. A., Silva, F. A., and Lagrota, V. (2026). Matrix protocol: a comprehensive systematic mapping study. *Journal of Cloud Computing*, 15(1):20.

Martins, J. A. P., Rego, P. A. L., Macêdo, J. A. F. d., Andrade, R. M. C., Bonfim, M. S., Ivo, R. F., Costa, V. L. R. d., Pacheco, R. P., and Silva, F. A. P. d. (2025). Protocolo matrix: Conceitos, arquitetura, aplicações e desafios. In *Jornada de Atualização em Informática 2025*. SBC, Porto Alegre, RS, Brasil.

Müller, M., Lee, J.-U., Frick, N., Stangier, L., Gurevych, I., and Metternich, J. (2021). Extracting problem related entities from production chats to enhance the data base for assistance functions on the shop floor. *Procedia CIRP*, 103:231–236.

Stocker, V., de Camargo, E. T., and De Bona, L. C. (2025). Uma investigação sobre tolerância a falhas no servidor de comunicação synapse. In *Workshop de Testes e Tolerância a Falhas (WTF)*, pages 57–70. SBC.

Performance Analysis and Refactoring of the Synapse Reference Server through Observability

Francisco A. A. Gomes[1], Francisco A. Silva[2], Michel Sales[1], Windson Viana[1], Fernando A. M. Trinta[1], Rossana M. de C. Andrade[1], José A. F. de Macêdo[1], Vinícius Lagrota[3], Rodrigo Pacheco[3], Paulo A. L. Rego[1]

[1]Universidade Federal do Ceará (UFC)

[2]Universidade Federal do Piauí (UFPI)

[3]Centro de Pesquisa e Desenvolvimento para a Segurança das Comunicações (CEPESC)

almada@crateus.ufc.br michelsb@ufc.br windson@virtual.ufc.br

{fernando.trinta,jose.macedo,paulo,rossana}@dc.ufc.br

faps@ufpi.edu.br {vinicius.1232,rodrigo.6874}@abin.gov.br

Abstract. *The Matrix protocol is a leading solution for decentralized, real-time communication, with Synapse as its primary reference server. Despite its wide adoption, systematic performance evaluations leveraging observability have been limited. This study instruments Synapse with distributed tracing, metrics collection, and network traffic analysis to characterize endpoint usage, identify bottlenecks, and guide architectural optimization. Our analysis revealed that synchronization, database contention, and lock management are the primary sources of latency, particularly around the* `/sync` *and message dispatch endpoints. Guided by observability insights, we applied targeted refactoring, including cache resizing, workload redistribution to specialized workers, bare-metal deployment, PostgreSQL tuning, and connection pooling. These interventions transformed Synapse into a highly scalable system, increasing throughput from 11 to 405 requests per second and reducing average response time from 4755 ms to 92 ms. The results demonstrate that observability not only diagnoses performance issues but can directly inform refactoring strategies to achieve substantial operational gains in complex, distributed communication platforms.*

1. Introduction

Synapse is the reference implementation of the Matrix protocol [Martins et al. 2025], an open standard designed for decentralized communication that enables real-time messaging, voice, and video across a federated network of servers. Developed in Python, it offers client–server and server–server APIs, supporting end-to-end encryption, and message synchronization, which makes performance and scalability central concerns. Beyond its technical design, Matrix has been adopted in both public and private sectors, including governmental organizations in Germany, France, Sweden, and Luxembourg, as well as within the free and open-source ecosystem through initiatives such as Mozilla, and KDE [Martins et al. 2026]. With more than 80 million reported users and active discussion within the IETF on interoperability standards, Matrix represents a significant case study in the deployment and evolution of decentralized communication infrastructures [Albrecht et al. 2024].

Synapse instances consist of multiple interdependent components that can be conceptualized as microservices, endowing the system with features of complex cloud-native applications such as high distribution, dynamic behavior, and unpredictability. This distributed nature increases the likelihood of failures and performance degradation, making system diagnosis particularly challenging, especially when many component instances operate concurrently [Dragoni et al. 2017]. Conventional monitoring solutions often fail to provide comprehensive visibility and cannot proactively detect deviations from expected behavior before they affect users.

Such challenges have driven the embrace of observability, an advanced form of monitoring intended to provide actionable insights into system behavior [Kosińska et al. 2023]. Observability refers to the ability to deduce the internal state of a complex system from its externally visible signals [Usman et al. 2022], consolidating metrics, logs, and traces into a coherent view of how requests traverse components, where time is spent, and which resources are under pressure. Unlike traditional monitoring, which often focuses on predefined indicators and coarse alarms, observability supports exploratory investigation, allowing operators to ask new questions as the system evolves and workloads shift. In practice, it enables administrators and developers to continuously compare the system's runtime behavior to its expected operation across the software lifecycle, from deployment and scaling decisions to incident response and postmortem analysis. As a result, observability promotes informed performance optimization, strengthens reliability by surfacing hidden dependencies and contention points, and supports proactive issue detection before degradations become visible to end users [Karumuri et al. 2021].

By combining observability techniques with load testing, we identified critical inefficiencies in Synapse's database management layer. The collected metrics and traces revealed that a substantial portion of request latency is not associated with endpoint-specific logic, but with recurring delays in acquiring database connections, scheduling queries, and coordinating concurrent access to shared state. In particular, Synapse relies on asynchronous data consistency mechanisms that, under concurrent workloads, frequently trigger locking at the persistence layer and amplify contention among worker threads and processes. This behavior creates cascading delays: lock acquisition stalls request handling, queued operations accumulate, and the system becomes increasingly sensitive to bursts of traffic and database growth. Consequently, throughput fails to scale with added concurrency, while response times increase sharply, indicating that the observed degradation is rooted in structural bottlenecks rather than sporadic anomalies.

To address these issues, the system requires architectural refactoring focused on reducing contention within the data layer. This may involve database tuning strategies, such as connection pool optimization as well as improvements in caching mechanisms within the application itself to minimize unnecessary database access. Such measures are essential to ensure that Synapse can evolve from a functionally correct system into a truly scalable and resilient platform. In this context, this study aims to address the following research questions:

RQ1 *How is observability leveraged to characterize the usage patterns of Synapse's endpoints?*

RQ2 *What is the impact of observability-driven insights on architectural refactoring aimed at improving Synapse's scalability?*

Overall, the study reinforces the role of observability not only as a diagnostic mechanism but as a strategic tool for continuous system evolution.

The remainder of this paper is organized as follows. Section 2 discusses related work on Matrix/Synapse and observability in distributed systems. Section 3 describes the methodology, including the instrumentation setup and the experimental procedure. Sections 4 and 5 present the performance analysis and bottleneck identification, followed by the optimization strategy and the impact assessment. Finally, Section 6 concludes the paper and outlines threats to validity and directions for future work.

2. Synapse Homeserver

Synapse is a server implementation of the Matrix protocol designed to support decentralized, secure, and low-latency communication among distributed users. It operates as the core component responsible for managing messages, room states, presence information, devices, and trust relationships across different servers. By avoiding reliance on a centralized infrastructure, Synapse enables independent organizations to run their own servers while still participating in a federated communication network [Li et al. 2023].

The architecture of Synapse is organized into well-defined layers, with a strong dependence on the persistence layer to ensure data durability and consistency. PostgreSQL is typically used as the primary database, storing large volumes of events, message histories, and room states. This design allows clients to retrieve information even after extended periods of disconnection, but it also makes the persistence layer a critical point, as it concentrates intensive read and write operations under high activity levels. One of the most performance-sensitive components of Synapse is the synchronization mechanism exposed through the /sync API. This endpoint is responsible for delivering incremental updates to clients, aggregating recent events, state changes, and relevant metadata. The process involves complex database queries, joins, and filtering operations, making system performance highly dependent on client usage patterns and the degree of concurrent access [Bilyatdinov et al. 2026].

To handle higher workloads, Synapse supports task distribution through specialized workers combined with load balancing strategies. This separation of responsibilities allows functions such as federation, synchronization, and event processing to be isolated across different processes. When used together with reverse proxies and connection pooling mechanisms, this approach helps reduce contention at the database layer and improves overall concurrency handling. Federation is another fundamental aspect of Synapse, enabling independent servers to exchange events and maintain shared rooms across different domains. While this capability significantly extends the reach of communication, it also introduces additional challenges, including variable latency, eventual consistency, and increased internal traffic. Asynchronous coordination between servers requires careful management to preserve global state coherence, particularly in geographically distributed environments [Li et al. 2023].

Finally, Synapse has been studied due to its operational complexity; however, the use of observability as a systematic approach for performance analysis remains limited. Instrumentation using metrics, logs, and traces enables a deeper understanding of system behavior, allowing the identification of recurring bottlenecks and the evaluation of different architectural configurations. These analyses provide a solid foundation for infrastructure tuning, parameter adjustment, and the continuous evolution of Synapse in

demanding production scenarios.

3. Related Work

Six papers were identified as having synergy with the objectives of this paper. Three papers focus on user distribution across rooms and servers [Jacob et al. 2019, Jacob et al. 2021, Li et al. 2023]. Jacob et al. [Jacob et al. 2019] evaluated the scalability of Matrix, a message-driven data synchronization middleware designed for federated platforms supporting decentralized near-real-time applications. The study offers a comprehensive understanding of performance and scalability characteristics. Jacob et al. [Jacob et al. 2021] explored the Matrix Event Graph (MEG) as a replicated data type for decentralized applications. The work validates MEG as a Convergent and Replicated Data Type (CRDT), ensuring strong eventual consistency in distributed systems. Li et al. [Li et al. 2023] analyzed the structure and security vulnerabilities of the Matrix network through in-depth exploratory research, focusing on homeservers, rooms, and users while exposing challenges related to decentralization, cybersecurity, and cryptographic protocols.

Three other papers emphasize security and privacy in Matrix [Müller et al. 2021, Riutort Grande 2021, Albrecht et al. 2023]. Muller et al. [Müller et al. 2021] examined the integration of informal communications into digital shop floor management systems, aiming to structure these exchanges for improved issue detection and resolution. Riutort et al. [Riutort Grande 2021] proposed a TOR-based privacy-preserving framework for IoT environments, enabling anonymous communication in devices with limited processing capacity. Albrecht et al. [Albrecht et al. 2023] revealed exploitable vulnerabilities in the Matrix protocol and the Element client, highlighting weaknesses in confidentiality and authentication guarantees when exposed to malicious servers. Regarding the use of Observability, some studies aim to contextualize the topic but do not provide specific information about its application in Synapse. For instance, Li et al. [Li et al. 2022] investigated microservices and observability, emphasizing the complexity of industrial microservice systems deployed in intricate cloud environments, where service instances are dynamically created and terminated. Their work highlights that microservice tracing and analysis represent a new big data challenge for software engineering, while also unveiling opportunities such as adaptive log sampling, data fusion for trace analysis, intelligent trace processing, and business intelligence derived from trace data. Similarly, Usman et al. [Usman et al. 2022] conducted a comprehensive study on distributed IT environments and microservices observability, aiming to identify challenges, requirements, best practices, and existing solutions for monitoring distributed systems. Moreover, the authors underscore the importance of organizational culture and individual mindsets in influencing the adoption of emerging observability tools and practices.

Table 1 compares the literature using five direct dimensions. *Matrix scope* states which layer each work targets (federation topology, room-level MEG replication, protocol/client security, or domain adoption). *Method* shows how evidence is obtained (large-scale measurement, formal/stochastic analysis, architectural design, integration, or practical exploitation). *Key signals/outputs* lists what each study delivers (e.g., skew/distribution patterns, SEC/CRDT and graph dynamics, threat surfaces and attack conditions, privacy trade-offs, or structured communication artifacts). *Main axis* labels the dominant concern (scalability, consistency, security, privacy, integration).

Unlike previous studies, which primarily focus on scalability concerns of the Ma-

trix protocol or explore security and privacy aspects, this work advances the state of the art by employing observability as a systematic method for performance evaluation in the Synapse reference server. Instead of limiting the analysis to protocol design or threat surfaces, we leverage observability signals to identify real usage patterns, reveal scalability constraints in database and concurrency management, and determine the exact conditions under which performance degradation occurs. This approach perspective not only exposes operational bottlenecks, but also provides concrete guidance for architectural refactoring and optimization that are not addressed in prior research.

Table 1. Related work comparison.

Ref.	Matrix scope	Method	Key signals / outputs	Main axis
[Jacob et al. 2019]	Federation-wide user/room/server distribution	Empirical measurement + large-scale analysis of federation structure	Load concentration patterns; room/server skew; scalability characterization	Scalability
[Jacob et al. 2021]	Matrix Event Graph (MEG) per room; replication and convergence	Formal analysis + stochastic modeling (Markov chains) of MEG dynamics	SEC/CRDT validation; graph width / forward-extremities behavior	Consistency
[Li et al. 2023]	Network ecosystem (homeservers, rooms, users) + security posture	Large-scale measurement (MatSpider) + exploratory security analysis	Distribution evidence; exposed threat surfaces and risks	Security
[Müller et al. 2021]	Matrix-based informal communication in industrial shop floor	Integration + extraction/structuring of communications for issue handling	Structured communication artifacts for detection/resolution workflows	Integration
[Riutort Grande 2021]	Privacy-preserving communication for constrained IoT (TOR-based)	Architecture proposal + feasibility discussion in constrained devices	Anonymity/privacy properties; overhead/constraints trade-offs	Privacy
[Albrecht et al. 2023]	Matrix protocol + Element client under malicious server assumptions	Practical protocol/client security analysis and exploitation scenarios	Breaks in confidentiality/authentication under adversarial conditions	Security
This work	Synapse reference homeserver under realistic workloads	Observability-driven performance evaluation + load testing	Metrics/logs/traces to pinpoint DB/concurrency bottlenecks; degradation conditions	Performance

4. Methodology

In this study we have used the Synapse v1.122.0. To address the research questions, we organized the study into five main stages. In the **first stage**, we conducted a load test to identify performance bottlenecks and instances of low scalability in the Synapse server. The **second stage** involved applying observability techniques (e.g., distributed tracing) to characterize the usage patterns of Synapse's endpoints and uncover the root causes

of performance degradation, aiming to address *RQ1*. In the **third stage**, we analyzed the collected traces and identified that the main source of latency originated from inefficiencies in the database layer. The **fourth stage** consisted of an architectural refactoring process, in which we adjusted configurations and introduced new components to improve performance and scalability. Finally, in the **fifth stage**, we conducted a new load test to evaluate the impact of these changes, providing evidence to answer *RQ2* regarding the effectiveness of observability-driven insights for improving Synapse's scalability.

4.1. Materials and Methods

In this study, Synapse was deployed on a Kubernetes cluster to evaluate its behavior in a cloud-native, microservices-based environment, where dedicated resources were used to minimize interference from external processes. Figure 1 shows an overview of all tools and Synapse clients involved. Prometheus[1] was integrated to collect time-series metrics from the Synapse application, providing detailed insights into database and cache performance. Synapse natively exports metrics related to database operations, events, and cache usage with built-in support for Prometheus, simplifying monitoring setup.

Complementing this, Kubeshark[2] was employed to capture and analyze real-time network traffic across pods and services, including ingress, egress, and internal communications for protocols such as HTTP, gRPC, Redis, and Kafka, without requiring any code instrumentation. This enabled the identification of the most frequently used endpoints and an understanding of service interaction patterns.

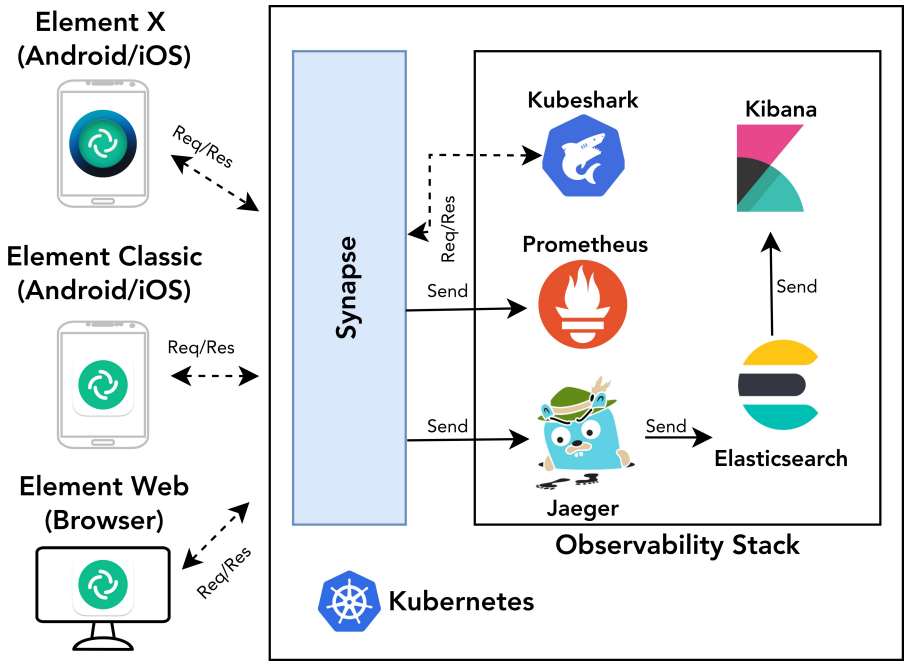


Figure 1. Observability Architecture for Synapse

In addition, Synapse includes instrumentation for distributed traces and offers native support for exporting them to Jaeger[3], enabling end-to-end tracking of requests across

[1] https://prometheus.io/
[2] https://www.kubeshark.co/
[3] https://www.jaegertracing.io/

multiple microservices and identification of the most time-consuming operations. All traces collected by Jaeger were stored in Elasticsearch[4], which provides efficient indexing and query capabilities, and visualized through Kibana[5] dashboards to facilitate interactive exploration and deeper comprehension of Synapse's internal behavior. The integrated use of Kubeshark, Jaeger, Prometheus, Elasticsearch, and Kibana enabled both metric and trace-level analysis of the system, supporting performance optimization and offering a robust framework for monitoring and understanding Synapse deployments in dynamic, distributed Kubernetes environments.

5. Workload Characterization and Baseline Analysis

Through Kubeshark, we monitored traffic within the Kubernetes cluster and create a dataset (made available in the Artifact Availability section) containing the requests, including their method, path, and timestamp. The traffic was generated by approximately 50 real users over the course of one week in an industry project using Synapse, whose name is omitted for confidentiality. The Table 2 presents the 20 most frequent requests.

Table 2. The 20 most frequent requests in the dataset.

Request	Count
GET /_matrix/client/v3/sync	441089
GET /_matrix/client/r0/sync	122327
GET /_matrix/media/r0/thumbnail/	48991
POST /_matrix/client/unstable/org.matrix.simpl...	25166
GET /_matrix/client/v1/media/thumbnail/	14585
GET /_matrix/client/versions	14085
PUT /_matrix/client/v3/rooms/typing	10117
POST /_matrix/client/r0/keys/query	7333
PUT /_matrix/client/r0/rooms/typing	5904
POST /_matrix/client/r0/rooms/read_markers	5596
POST /_matrix/client/v3/rooms/read_markers	5554
GET /_matrix/client/r0/devices/	5386
POST /_matrix/client/v3/keys/query	4377
POST /_matrix/client/v3/rooms/receipt/m.read/	4038
GET /_matrix/client/r0/devices	3496
PUT /_matrix/client/v3/rooms/send	3144
POST /_matrix/push/v1/notify	2685
PUT /_matrix/client/v3/user/account_data/im.ve...	2250
GET /_matrix/client/r0/pushers	2142
PUT /_matrix/client/r0/rooms/send	2048

From Table 2, we observe the relevance of the `/Sync` functionality to the operation of communication applications that rely on Synapse. This functionality represents one of the core components of the Matrix protocol, as it ensures that clients remain updated with the state changes occurring in the rooms they participate in. The `/Sync` mechanism enables clients to receive the latest messages sent by other users, join and leave events, presence updates, and additional information required to maintain a consistent and up-to-date timeline between server and client. Its operation resembles a polling strategy with a maximum timeout (30s), where the client periodically issues requests to the server

[4]`https://www.elastic.co/elasticsearch`
[5]`https://www.elastic.co/kibana`

in order to retrieve recent updates. Table 2 highlights three different *Sync* endpoints (R0, V3, and MSC3575), with the latter corresponding to the most frequent `POST` request (i.e., `/unstable/org.matrix.simplified_msc3575`). These versions represent distinct synchronization implementations, depending on the Matrix client adopted (e.g., Element relies on R0, Element Web on V3, and Element X on the simplified MSC3575).

6. Performance Analysis and Bottleneck Identification

Before conducting any synthetic stress tests, we examined system health using observability-driven metrics collected under normal runtime conditions. One of the most revealing indicators was the average time spent waiting for a database connection, collected by Prometheus, where the main Synapse process exhibited peaks of up to 27.9 ms. Since acquiring a database connection is among the very first steps of any operation, such delays inevitably cascade through the remaining execution stages, degrading performance across the board. Similarly, high database scheduling time quantiles exposed by Synapse and collected by Prometheus, representing the time required for database queries to be scheduled, peaked at 473 ms in the main process. These values provide strong evidence that responsiveness is being hindered primarily by contention at the database layer rather than by application-level logic.

To assess how this latent bottleneck manifests under user concurrency, we executed a load test using Locust[6], simulating repeated message-sending operations. The experimental machine is equipped with an 8-core Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz, 32 GB of RAM, and 100 GB of disk space. The baseline execution, performed without any tuning or architectural refinement, revealed a concerning fragility: with only 20 concurrent users, the system sustained a throughput of 11 requests per second, while the average response time escalated to 4755 milliseconds, a value fundamentally incompatible with real-time interaction requirements.

These observations indicate that the performance degradation is structural rather than incidental. Instead of applying incremental fixes without direction, we conducted a deeper inspection of the execution model, which showed that the main process frequently faced excessive waiting time and lock contention among worker threads distributed across Kubernetes pods. In highly threaded environments, competition for the Global Interpreter Lock (GIL) further constrained parallelism and introduced latency. Message dispatching also triggered a disproportionate number of interactions with Redis and the relational database, often issuing multiple queries for a single message. Although persistent database connections were intended to reduce overhead, under load they increased contention. As the database state grew, query execution times degraded proportionally, reinforcing the perception that Synapse aggressively consumes database resources and slows down as stored data accumulates.

7. Observability-Driven Architectural Refactoring

Having confirmed through observability metrics, execution traces, and empirical load testing that database contention was the dominant constraint on scalability, the next phase focused on a comprehensive architectural refactoring rather than superficial tuning. The traces were particularly valuable in pinpointing which operations contributed most to latency, providing granular visibility that complemented the high-level metrics. This archi-

[6] `https://locust.io/`

tectural approach aimed to systematically redistribute workload, optimize resource utilization, and reduce contention across the system.

Key interventions included resizing Synapse's internal cache structures to minimize redundant database lookups and reduce read amplification. Many responsibilities previously handled by the main process were offloaded to specialized workers, such as background workers, media handlers, application bridges, user directory processors, pushers, event persisters, and generic workers configured in parallel. By isolating these components into dedicated execution units, contention around shared resources was significantly reduced. Both the main process and the worker pool were deployed as bare-metal services managed via `systemd`, removing overhead from container orchestration layers. Inter-process communication was optimized using UNIX domain sockets for replication, replacing HTTP and Redis intermediations, and hostname resolution was eliminated from the critical path by directly using IP addresses to avoid DNS-induced stalls.

On the persistence layer, PostgreSQL was carefully tuned to sustain high levels of write concurrency, while PgBouncer was deployed as a lightweight connection-pooling proxy in front of the database. This setup enabled efficient reuse of established sessions, mitigating connection storms under bursty workloads and lowering transaction setup overhead, an effect that is particularly noticeable for short-lived interactions that would otherwise spend a disproportionate amount of time on connection negotiation. In addition to stabilizing database access patterns, the configuration helped smooth resource consumption at the database tier (e.g., backends, memory, and lock management), improving overall predictability during peak synchronization periods. Furthermore, synchronization traffic was load-balanced across the fleet of generic workers so that update propagation remained evenly distributed, avoiding hotspots where requests could concentrate on a single worker or thread of execution. As a result, the system reduced queue buildup during propagation waves and maintained steadier throughput, even when multiple clients issued concurrent updates and the persistence layer was under sustained pressure.

8. Impact Assessment

This section assesses the impact of the proposed refactoring by comparing Synapse's performance before and after the observability-driven interventions, focusing on scalability, throughput, and response time under load, which capture both system efficiency and user-perceived latency. After the Synapse optimization, we rerun the same load test, the transformation was substantial. With 20 concurrent users, throughput increased from 11 to 405 requests per second, while the average response time dropped from 4755 ms to just 92 ms. Execution traces collected after optimization confirmed that the operations that previously dominated latency no longer appeared as persistent outliers, validating that the architectural interventions effectively neutralized the critical bottlenecks. Additionally, the average time spent waiting for a database connection dropped to 1.06 ms, and the database scheduling time quantiles exposed by Synapse reached peaks of 10.6 ms. Thus, by also leveraging these metrics, it was possible to verify a significant improvement in database management indicators.

These results indicate that the performance bottlenecks were rooted in the system's architecture rather than being occasional anomalies. They also show how observability, through both metrics and traces, can effectively support architectural decisions. Grounded in concrete evidence, the changes were focused and meaningful, turning an execution pipeline that initially struggled under load into a distributed, scalable, and high-

throughput messaging core capable of meeting the demands of modern communication workloads.

8.1. Tracing Evaluation

Using the Element Web, the instrumentation of Synapse using Jaeger was carried out in a controlled experiment where two users, to simulate message exchange, exchanged approximately two hundred messages in a local Matrix instance. This setup ensured a realistic but reproducible workload, generating sufficient activity to expose the internal dynamics of the server. By leveraging spans already instrumented in the Synapse codebase, we collected a comprehensive set of operations executed during message handling, synchronization, and background processing. Two rounds of measurement were performed: one before and one after the refactoring process. This comparison allowed for a quantitative assessment of performance improvements and identification of which components benefited most from the architectural and tuning changes.

While the raw list of spans is extensive, grouping them by functionality allows us to identify the most critical areas of overhead and the types of work performed by the server. Operations are divided into categories such as REST services, caching and synchronization, database queries, and concurrency control. For each group, we report average execution times before and after refactoring, along with the factor by which the execution time decreased, and discuss their implications for performance and scalability. This layered view highlights where bottlenecks occurred prior to refactoring and how the optimization efforts contributed to more efficient processing.

8.1.1. REST Services: Entry Points for Client Requests

This group encompasses the main operations directly exposed to external clients through HTTP requests (see Table 3). Each servlet handles a different type of request, from synchronization of timelines to sending new events or updating profile information. Since these operations mark the boundary between the client and the server, their efficiency directly determines the responsiveness perceived by end users.

Table 3. REST service operations before and after refactoring.

Operation	Before (ms)	After (ms)	Times Faster
SyncRestServlet (with timeout)	10,582	10,422	1.02x
SyncRestServlet (without timeout)	56.0	39.1	1.43x
RoomSendEventRestServlet	282.7	165.3	1.71x
ReadMarkerRestServlet	216.3	47.2	4.58x
RoomTypingRestServlet	22.8	14.2	1.61x

The results confirm that the `SyncRestServlet` continues to dominate system activity. This servlet is requested by the Sync routes R0 and V3, as indicated in Table 2. With an average execution of 10 s (with timeout) and 56 ms (without timeout), this operation reflects the inherent cost of maintaining long-poll connections at the `/sync` endpoint. The span duration includes both waiting for events and preparing responses, which explains why it is significantly longer than other endpoints. After the refactoring, this servlet did not show any meaningful improvement in the long-poll (with timeout)

variant, as its duration is mostly determined by the expected waiting time for new events rather than by server processing. However, in the non-blocking variant (without timeout), execution time decreased by approximately 1.43 times, indicating that the refactoring effectively reduced overhead related to message preparation and event processing.

Other endpoints benefited more substantially. The `RoomSendEventRestServlet`, responsible for accepting and propagating new events, became 1.71 times faster, while the `ReadMarkerRestServlet`, responsible for updating the read position of a user in a room, improved 4.58 times, reflecting a substantial reduction in database and cache processing overhead. Similarly, the `RoomTypingRestServlet`, responsible for notifying that a user is typing in a room, became 1.61 times faster, which can be attributed to lighter synchronization mechanisms and reduced contention on shared state.

8.1.2. Caching and Synchronization Processes

In addition to the REST-level spans, several operations represent the internal mechanisms that support synchronization (see Table 4). These include the use of caches, waiting loops for new events, and functions that prepare room data for delivery to clients. Because the sync endpoint is invoked so frequently, these operations strongly influence system scalability.

Table 4. Caching and synchronization-related operations before and after refactoring.

Operation	Before (ms)	After (ms)	Times Faster
ResponseCache[sync].calculate (with timeout)	10,142	10,052	1.01x
wait_for_events (with timeout)	10,070	10,032	1.01x
ResponseCache[sync].calculate (without timeout)	49.55	36.2	1.37x
wait_for_events (without timeout)	48.2	33.2	1.45x

The `ResponseCache[sync].calculate` span averages almost the same cost as the sync servlet itself, with and without timeout. This indicates that computing cache entries for sync responses is nearly as expensive as performing the synchronization without caching. In practice, this suggests that cache misses are extremely costly, and that further optimizations in response reuse could reduce load considerably. The `wait_for_events` operation, with an average of 10 s (with timeout), represents the blocking time during long-polling. This is not pure CPU time but waiting on external triggers, and yet it still ties up system resources and contributes to the high latency. In the without timeout case, the improvements were $1.37\times$ and $1.45\times$, respectively, indicating that the refactoring enhanced the caching and synchronization process.

8.1.3. Database Operations

Database calls form the backbone of Synapse, as most features require reading or updating state in persistent storage (see Table 5). The traced operations reveal a wide range of performance characteristics. Some queries, especially those related to account data, take more than one second on average, while many others execute in a few milliseconds.

Table 5. Representative database operations before and after refactoring.

Operation	Before (ms)	After (ms)	Times Faster
db.has_completed_background_updates	23.2	4.2	5.52x
db.get_account_data_for_room_and_type	17.1	7.8	2.19x
db.simple_select_one	7.3	5.4	1.35x
db.is_server_admin	17.0	8.2	2.07x
db.connection	5.3	2.1	2.52x
db.query	3.1	1.2	2.58x

The most time-consuming, `db.has_completed_background_updates`, decreased from 23.2 ms to 4.2 ms, becoming approximately 5.5 times faster. Similarly, `db.get_account_data_for_room_and_type` improved from 17.1 ms to 7.8 ms (2.2 times faster), and `db.is_server_admin` decreased from 17.0 ms to 8.2 ms (2.1 times faster). Smaller operations also benefited: `db.simple_select_one` went from 7.3 ms to 5.4 ms (1.35 times faster), `db.connection` decreased from 5.3 ms to 2.1 ms (2.5 times faster), and generic queries (`db.query`) improved from 3.1 ms to 1.2 ms (2.6 times faster). These results indicate that the refactoring substantially enhanced the responsiveness of database operations. Both high-cost and routine queries execute faster, contributing to lower latency across the system and improving the overall efficiency of frequently executed tasks.

8.1.4. Locking and Concurrency Management

Concurrency management is critical in distributed systems like Synapse. Locks ensure that only one process modifies a shared resource at a time, but lock contention can delay operations and reduce throughput. Traced spans related to locking provide insight into where contention arises and how the refactoring affected performance.

Table 6. Locking and concurrency management operations before and after refactoring.

Operation	Before (ms)	After (ms)	Times Faster
WaitingLock.lock	181.6	112.3	1.62x
db.drop_lock	7.2	5.1	1.41x
WaitingLock.waiting_for_lock	10.8	5.6	1.93x
db.try_acquire_read_write_lock	10.5	3.2	3.28x

The data shows that acquiring locks is the most time-consuming operation, with `WaitingLock.lock` decreasing from 181.6 ms to 112.3 ms (1.62 times faster) after refactoring. Other lock-related operations also benefited: `db.drop_lock` improved 1.41 times, `WaitingLock.waiting_for_lock` became 1.93 times faster, and `db.try_acquire_read_write_lock` showed the largest improvement at 3.28 times faster. Overall, the refactoring effectively reduced contention and improved concurrency management.

9. Final Considerations

This study showed that observability, combining metrics and distributed traces, is essential to identify structural bottlenecks in Synapse, particularly in synchronization, database access, and concurrency management. Traces were crucial in pinpointing latency-critical operations, guiding architectural refactoring that transformed the system into a scalable, high-throughput messaging core. Optimizations such as cache resizing, workload redistribution to specialized workers, bare-metal deployment, UNIX socket communication, PostgreSQL tuning, and PGBouncer integration substantially improved performance, with throughput increasing from 11 to 405 requests per second and average response time dropping from 4755 ms to 92 ms.

9.1. Research Questions

Moreover, to directly address the research questions: for **RQ1**, observability, combining metrics, traces, and network traffic analysis, enabled us to identify that the `/sync` endpoint dominates system activity and contributes disproportionately to latency. Tracing revealed which internal operations, such as cache calculations and event waiting, consume the most time, while metrics highlighted bottlenecks in database access and lock contention. This demonstrates that observability can effectively characterize endpoint usage and pinpoint critical performance hotspots.

For **RQ2**, leveraging insights from traces and metrics allowed us to identify structural bottlenecks and design targeted interventions. Architectural refactoring included resizing caches, offloading work to specialized workers, deploying main and worker processes as bare-metal services, optimizing inter-process communication via UNIX sockets, tuning PostgreSQL, and introducing PGBouncer for connection pooling. These measures, guided by observability data, significantly improved throughput and reduced response times, confirming that observability can effectively inform scalability-focused architectural decisions.

9.2. Threats to Validity and Future Work

Despite our analysis, some threats to validity remain. Load tests and traces were conducted in a controlled Kubernetes setup with limited users, which may not reflect large-scale deployments. However, the identified bottlenecks and optimizations target structural limitations expected to persist under higher loads. Performance gains are specific to the Synapse version and configuration used, and external factors like network latency, disk I/O, or background workloads could affect the observed results. Future work includes exploring adaptive caching, finer-grained concurrency control, predictive observability for proactive performance management, and scalable worker orchestration to further enhance responsiveness and throughput in large Matrix deployments.

Artifact Availability

The dataset with the requests is available at: https://doi.org/10.5281/zenodo.19454911

References

Albrecht, M. R., Celi, S., Dowling, B., and Jones, D. (2023). Practically-exploitable cryptographic vulnerabilities in matrix. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 164–181. IEEE.

Albrecht, M. R., Dowling, B., and Jones, D. (2024). Device-oriented group messaging: a formal cryptographic analysis of matrix'core. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 2666–1685. IEEE.

Bilyatdinov, K., Kiselev, S., and Petukhova, A. (2026). Information security model for messengers based on matrix protocol. *International Journal of Open Information Technologies*, 14(1):102–108.

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., and Safina, L. (2017). Microservices: yesterday, today, and tomorrow. *Present and ulterior software engineering*, pages 195–216.

Jacob, F., Beer, C., Henze, N., and Hartenstein, H. (2021). Analysis of the matrix event graph replicated data type. *IEEE access*, 9:28317–28333.

Jacob, F., Grashöfer, J., and Hartenstein, H. (2019). A glimpse of the matrix (extended version): Scalability issues of a new message-oriented data synchronization middleware. *arXiv preprint arXiv:1910.06295*.

Karumuri, S., Solleza, F., Zdonik, S., and Tatbul, N. (2021). Towards observability data management at scale. *ACM SIGMOD Record*, 49(4):18–23.

Kosińska, J., Baliś, B., Konieczny, M., Malawski, M., and Zieliński, S. (2023). Towards the observability of cloud-native applications: The overview of the state-of-the-art. *IEEE Access*.

Li, B., Peng, X., Xiang, Q., Wang, H., Xie, T., Sun, J., and Liu, X. (2022). Enjoy your observability: an industrial survey of microservice tracing and analysis. *Empirical Software Engineering*, 27:1–28.

Li, H., Wu, Y., Huang, R., Mi, X., Hu, C., and Guo, S. (2023). Demystifying decentralized matrix communication network: Ecosystem and security. In *2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 260–267. IEEE.

Martins, J. A., Rego, P. A., de Macêdo, J. A., Silva, F. A., and Lagrota, V. (2026). Matrix protocol: a comprehensive systematic mapping study. *Journal of Cloud Computing*, 15(1):20.

Martins, J. A. P., Rego, P. A. L., Macêdo, J. A. F. d., Andrade, R. M. C., Bonfim, M. S., Ivo, R. F., Costa, V. L. R. d., Pacheco, R. P., and Silva, F. A. P. d. (2025). Protocolo matrix: Conceitos, arquitetura, aplicações e desafios. In *Jornada de Atualização em Informática 2025*. SBC, Porto Alegre, RS, Brasil.

Müller, M., Lee, J.-U., Frick, N., Stangier, L., Gurevych, I., and Metternich, J. (2021). Extracting problem related entities from production chats to enhance the data base for assistance functions on the shop floor. *Procedia CIRP*, 103:231–236.

Riutort Grande, P. (2021). How p2p framework can help mitigate trust and security risks of iot applications. *Procedia CIRP*.

Usman, M., Ferlin, S., Brunstrom, A., and Taheri, J. (2022). A survey on observability of distributed edge "&" container-based microservices. *IEEE Access*, 10:86904–86919.