CEPESC

Centro de Pesquisa e Desenvolvimento para a Segurança das Comunicações

ABIN

Sumário

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 estas 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 i^{th} 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.

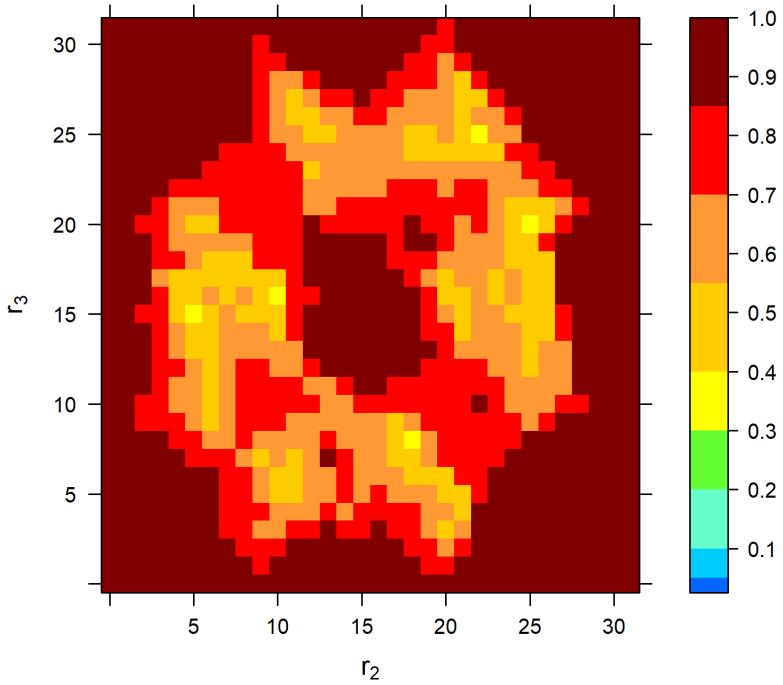


Figure 1. The biases were obtained for 3 rounds of ChaCha using rotations $r_1 = r_4 = 0$ and varying all values for r_2 and r_3. The color of the figure indicates the maximum absolute bias obtained for each combination of rotations. These are very poor results since that all biases are close to 1.

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

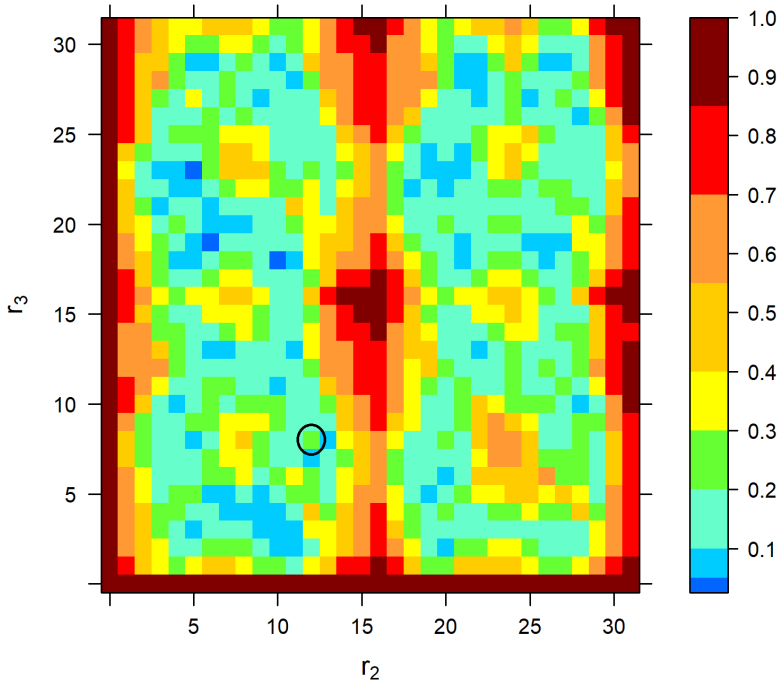


Figure 2. The biases we obtained for 3 rounds of ChaCha using rotations $r_1 = 16$ and $r_4 = 7$ and varying all values for r_2 and r_3. The color of the figure indicates the maximum absolute bias obtained for each combination of rotations. The value obtained for the original ChaCha is depicted inside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

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 i^{th} 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].

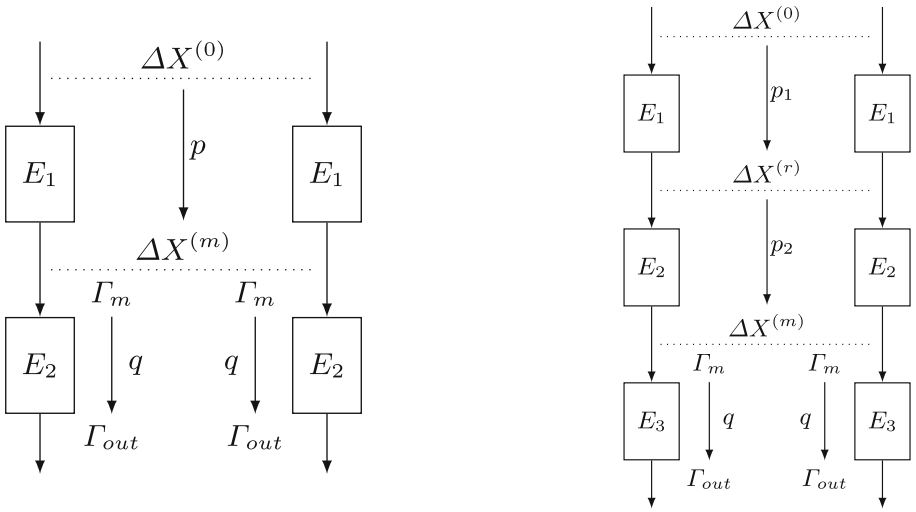


Fig. 1. A classical differential-linear distinguisher (on the left) and a differential-linear distinguisher with experimental evaluation of the correlation p_2 (on the right).

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 5^{th} 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.

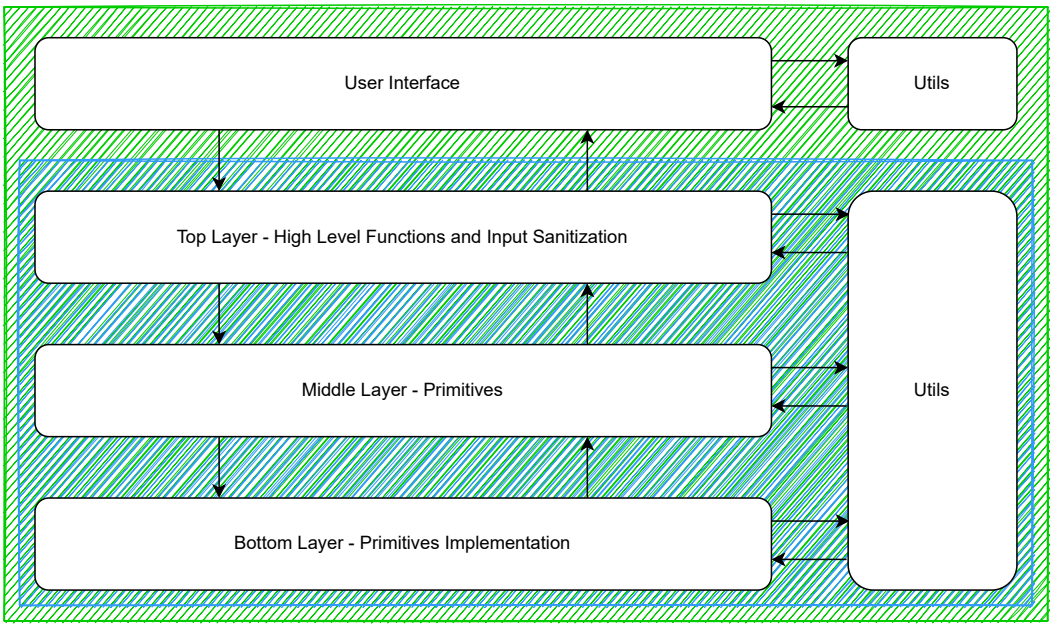


Figure 1. libharpia's architecture.

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.

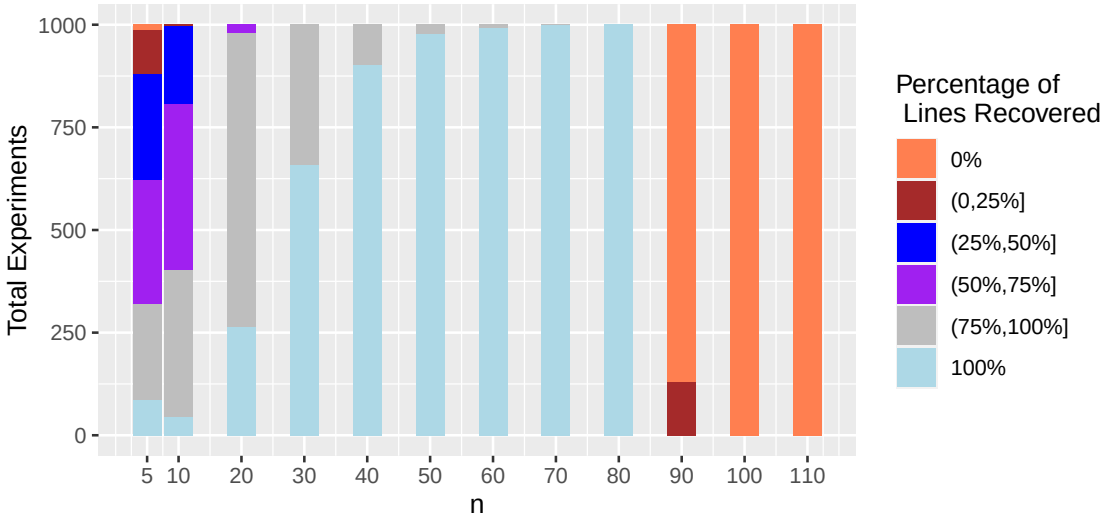
1. Compute $p^{-1} \mod q$and create the matrix L from eq.(3).
2. Run the BKZ algorithm on L and get L_{red}.
3. Use L_{red} to create a submatrix L^*_{red} with entries corresponding to the first n columns of L_{red}.
4. For each line f of the matrix F, check whether or not f is contained in one of 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.

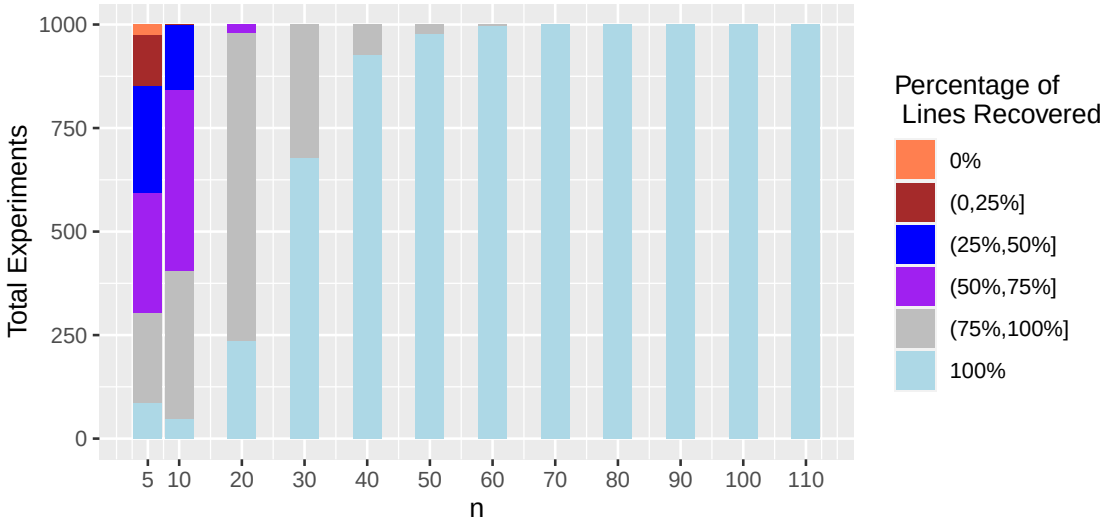
Proposition 3. *Let F, H be a corresponding private and public key pair for the matrix NTRU system with parameters n, p, q and let E be a ciphertext associated with the encryption of a message M such that the decryption process of E using F correctly returns M. Let F^* be a matrix formed by permuting 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

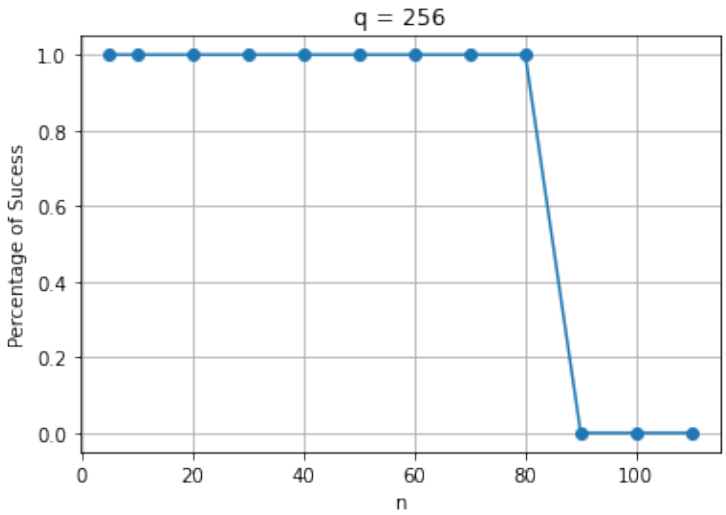
$$C_2 = F_p^*A_2 = F_p^*F^*M \mod p = M.$$

□

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.

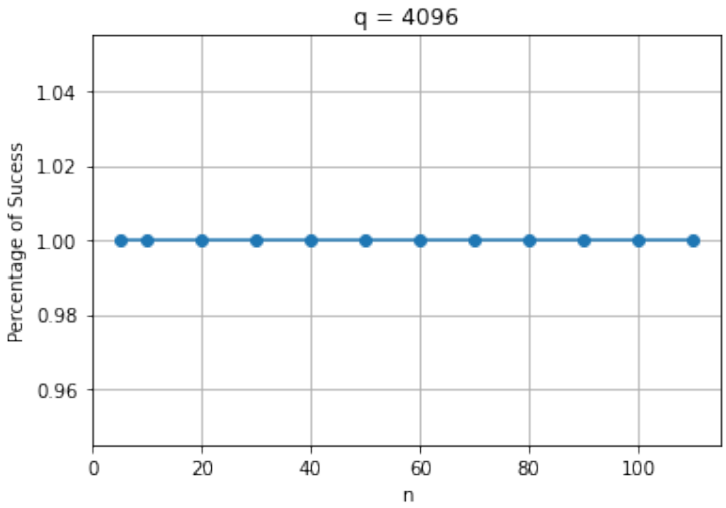


$q = 4096$) we can recover the true message in less than one minute. For $q = 256$ the attack works for dimension until $n = 80$ and for $q = 4096$ it works for dimensions up to 100. Since [Nayak et al. 2012] a matrix NTRU of dimension n with an NTRU of dimension n^2, this clearly shows that matrix NTRU 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.

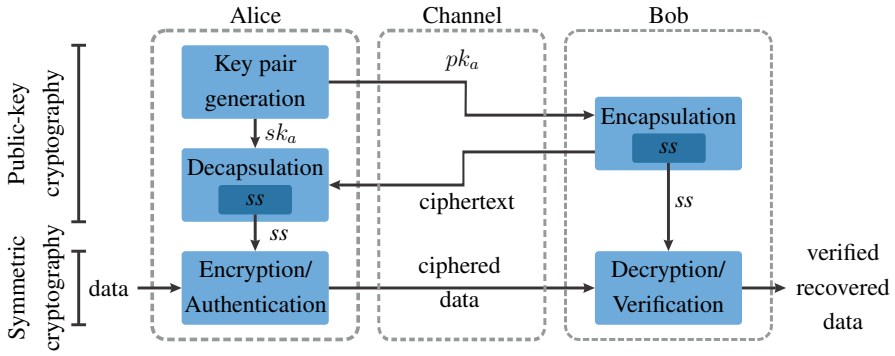


Figure 1. Block diagram illustrating the key agreement between parties and the encryption/decryption process. Note that pk_a, sk_a, and ss stand for public key, private key, and shared secret, respectively.

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.

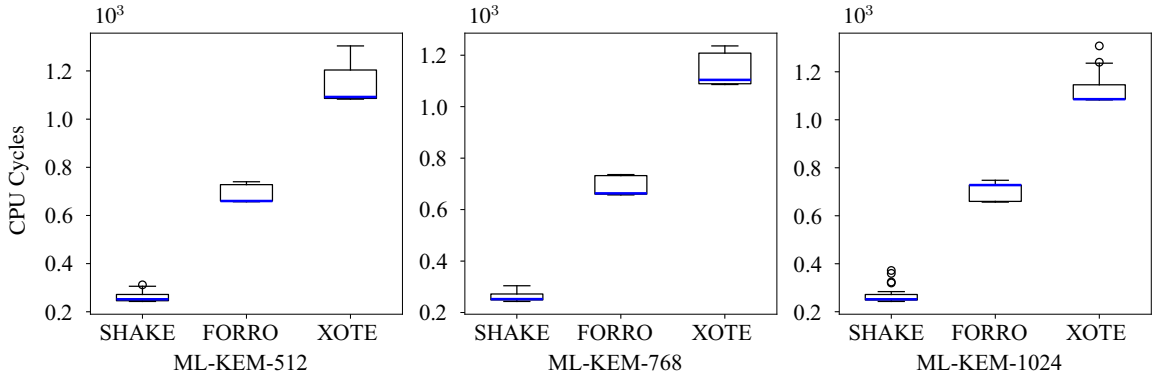


Figure 2. Boxplot of XOF-absorb execution time.

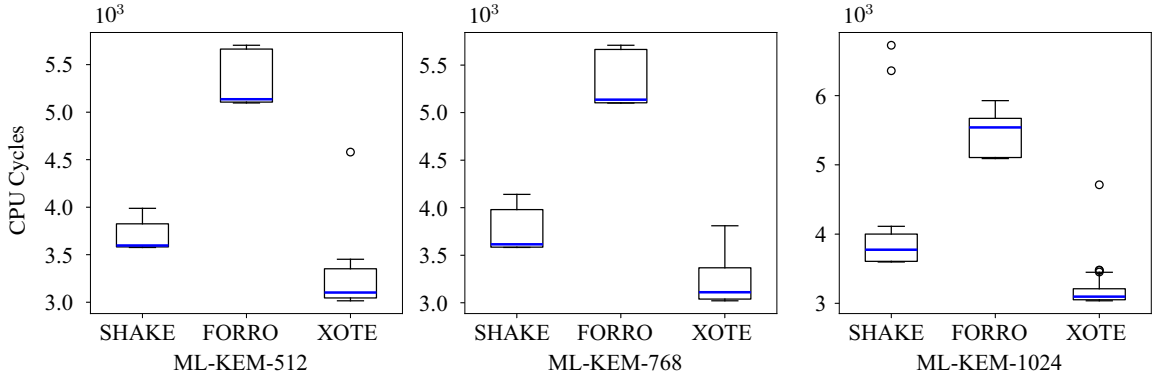


Figure 3. Boxplot of XOF-squeeze execution time.

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.

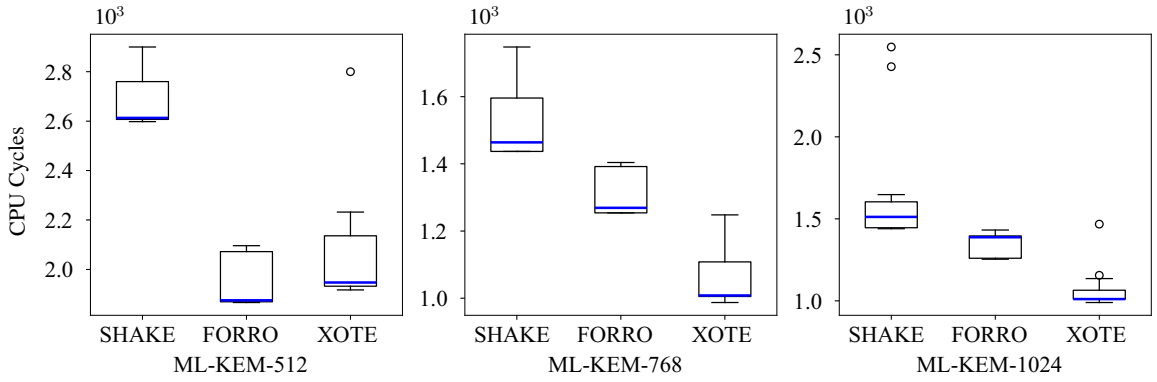


Figure 4. Boxplot of PRF execution time.

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.

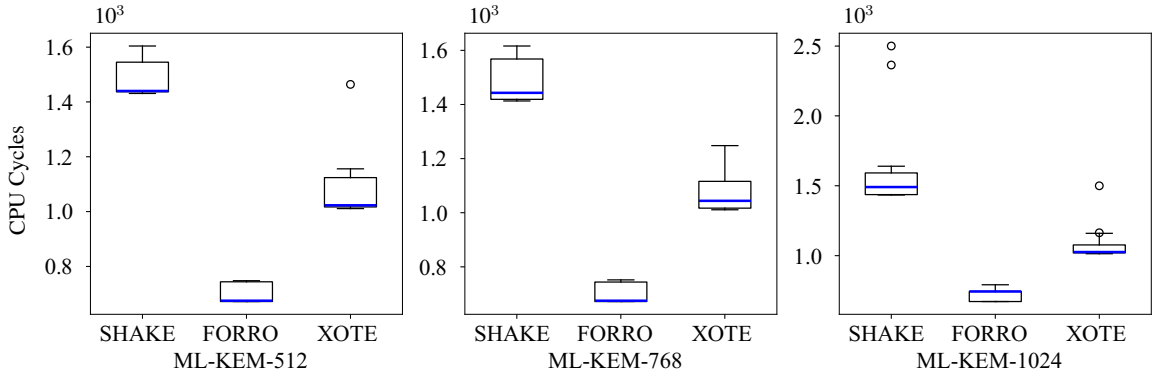


Figure 5. Boxplot of KDF execution time.

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.

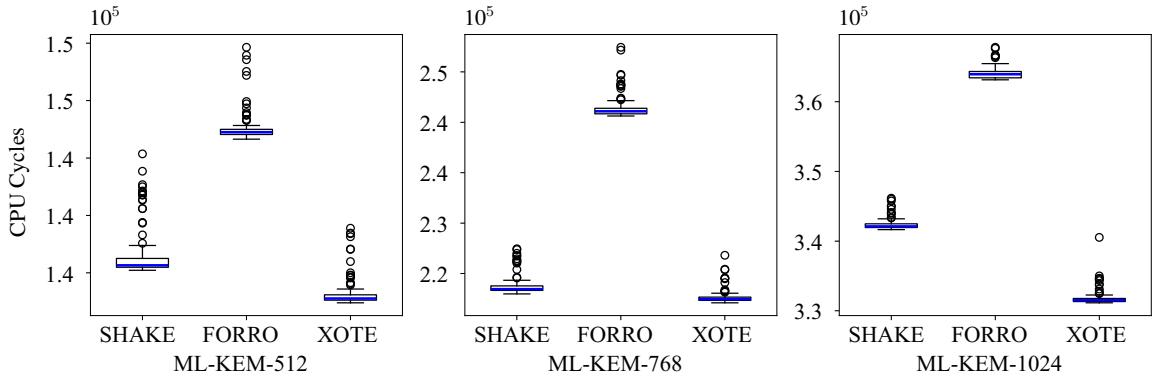


Figure 6. Boxplot of key pair generation execution time.

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.

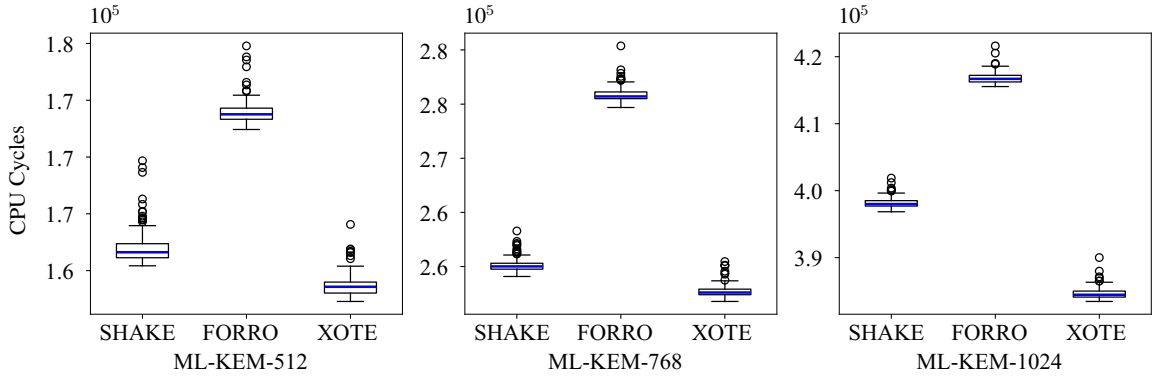


Figure 7. Boxplot of encapsulation execution time.

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.

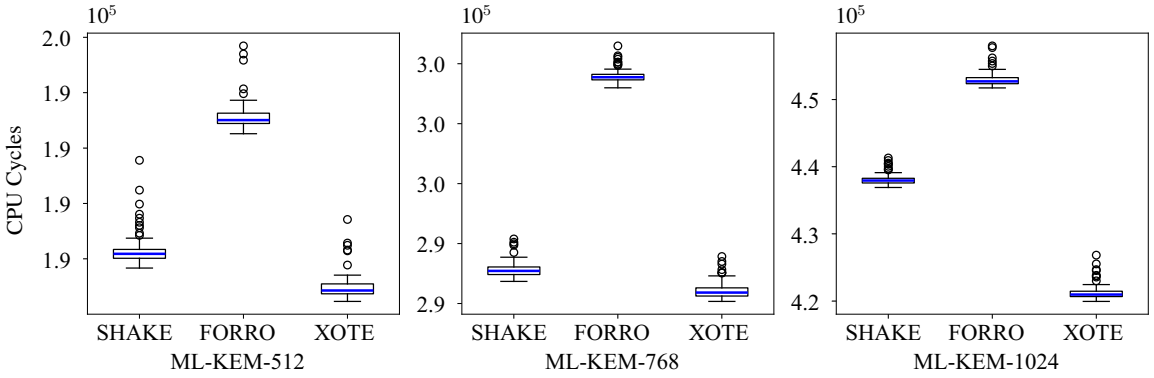


Figure 8. Boxplot of decapsulation execution time.

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.

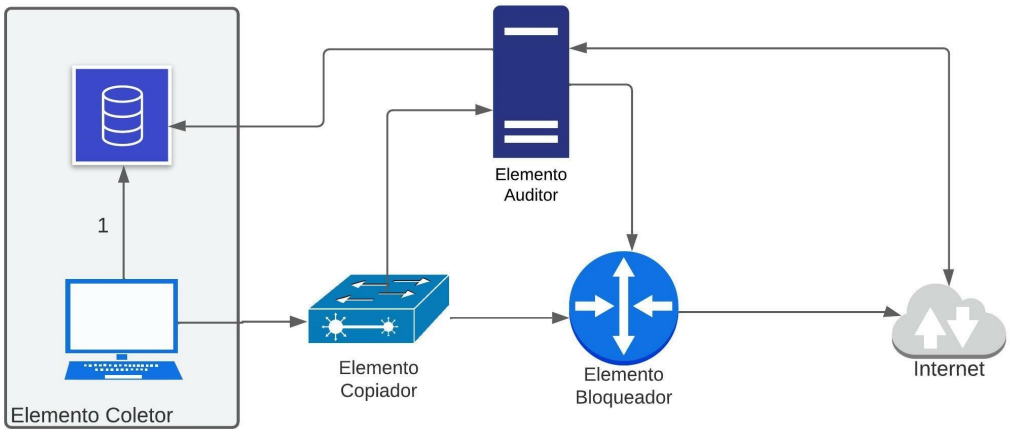


Figura 1. Arquitetura PARDED proposta (Figura do Autor)

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.

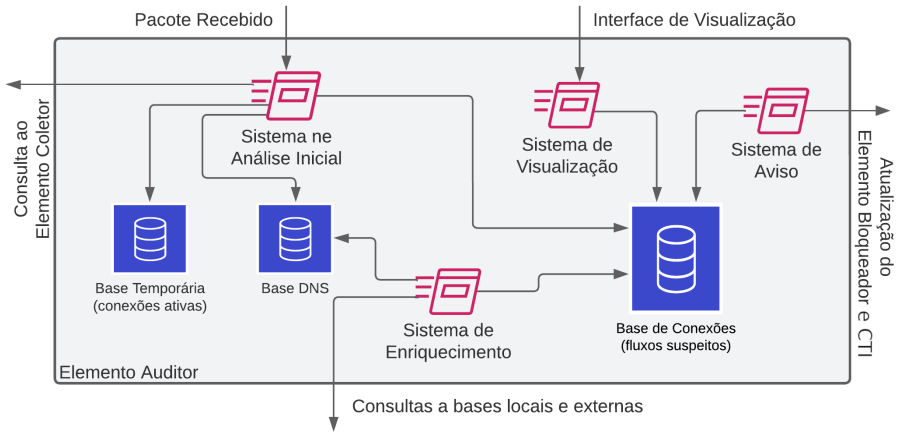


Figura 2. Elemento Auditor da estrutura PARDED (Figura do Autor)

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.