UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS REITOR JOÃO DAVID FERREIRA LIMA
PROGRAMA DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Daniel Boso

ESTUDO DE ALGORITMOS CRIPTOGRÁFICOS PÓS-QUÂNTICOS E SEUS CUSTOS

Florianópolis, Santa Catarina – Brasil
2021

Daniel Boso

ESTUDO DE ALGORITMOS CRIPTOGRÁFICOS PÓS-QUÂNTICOS E SEUS CUSTOS

Trabalho de Conclusão de Curso submetido ao Programa de Graduação em Ciências da Computação da Universidade Federal de Santa Catarina para a obtenção do Grau de Bacharel em Ciências da Computação.
Orientador(a): Paulo Manoel Mafra, Dr.
Coorientador(a): Jerusa Marchi, Dra.

Florianópolis, Santa Catarina – Brasil
2021

Notas legais:

Não há garantia para qualquer parte do software documentado. Os autores tomaram cuidado na preparação desta tese, mas não fazem nenhuma garantia expressa ou implícita de qualquer tipo e não assumem qualquer responsabilidade por erros ou omissões. Não se assume qualquer responsabilidade por danos incidentais ou consequentes em conexão ou decorrentes do uso das informações ou programas aqui contidos.

Catalogação na fonte pela Biblioteca Universitária da Universidade Federal de Santa Catarina.
Arquivo compilado às 18:46h do dia 23 de março de 2022.

Daniel Boso
Estudo de algoritmos criptográficos pós-quânticos e seus custos / Daniel Boso; Orientador(a), Paulo Manoel Mafra, Dr.; Coorientador(a), Jerusa Marchi, Dra. – Florianópolis, Santa Catarina – Brasil, 30 de março de 2019.
97 p.

Trabalho de Conclusão de Curso – Universidade Federal de Santa Catarina, INE – Departamento de Informática e Estatística, CTC – Centro Tecnológico, Programa de Graduação em Ciências da Computação.

Inclui referências

1. Criptografia, 2. Criptografia Pós-Quântica, 3. Criptografia baseada em código, 4. Criptografia baseada em hash, 5. Criptografia baseada em reticulado, 6. Criptografia polinomial multivariada, I. Paulo Manoel Mafra, Dr. II. Jerusa Marchi, Dra. III. Programa de Graduação em Ciências da Computação IV. Estudo de algoritmos criptográficos pós-quânticos e seus custos

CDU 02:141:005.7

Daniel Boso

ESTUDO DE ALGORITMOS CRIPTOGRÁFICOS PÓS-QUÂNTICOS E SEUS CUSTOS

Este(a) Trabalho de Conclusão de Curso foi julgado adequado(a) para obtenção do Título de Bacharel em Ciências da Computação, e foi aprovado em sua forma final pelo Programa de Graduação em Ciências da Computação do INE – Departamento de Informática e Estatística, CTC – Centro Tecnológico da Universidade Federal de Santa Catarina.

Florianópolis, Santa Catarina – Brasil, 30 de março de 2019.

Renato Cislaghi, Dr.
Coordenador(a) do Programa de
Graduação em Ciências da Computação

Banca Examinadora:

Paulo Manoel Mafra, Dr.
Orientador(a)
Universidade Federal de Santa
Catarina – UFSC

Jerusa Marchi, Dra.
Coorientador(a)
Universidade Federal de Santa
Catarina – UFSC

Eduardo Inácio Duzzioni, Dr.
Universidade Federal de Santa
Catarina – UFSC

Ricardo Felipe Custódio, Dr.
Universidade Federal de Santa
Catarina – UFSC

Este trabalho é dedicado ao Grupo de Pesquisa em Computação Quântica da UFSC e para todos a quem este trabalho possa ajudar de alguma forma.

AGRADECIMENTOS

Gostaria de agradecer primeiramente meus pais e minha namorada, Djulian, por todo o incentivo e apoio. Não posso esquecer também de todos os familiares que me incentivaram nestes anos de graduação. Meus agradecimentos ao Grupo de Pesquisa em Computação Quântica da UFSC por todo o aprendizado e também para meus orientadores, Jerusa Marchi e Paulo Manoel Mafra por proporem o tema e também por todos os conselhos e ajuda recebida ao desenvolver este trabalho.

Meus agradecimentos vão também para meus parentes de Florianópolis, pelo convite de morar na casa deles durante toda a graduação. Agradecimentos também aos grandes amigos que conheci na UFSC, principalmente ao Adan, Edison e Wesley por toda a parceria e a todos os colegas de curso.

“Mas qual é a vantagem da mensagem cifrada sem a cifra?”
O Vale do Medo, Arthur Conan Doyle

RESUMO

Apresenta-se um estudo dos algoritmos criptográficos pós-quânticos e seus custos através do acompanhamento do processo de padronização que o NIST está realizando. Com isso, conseguiu-se realizar um estudo de alguns algoritmos que estão em fase de padronização e que já passaram das primeiras etapas eliminatórias. Este estudo visou entender como estes algoritmos funcionam e seus custos. O processo do NIST ainda não terminou, mas já se vê melhorias em alguns algoritmos, onde os criadores conseguiram aumentar a eficiência como também diminuir o tamanho das chaves e assinatura.

Palavras-chaves: Criptografia. Criptografia Pós-Quântica. Criptografia baseada em código. Criptografia baseada em hash. Criptografia baseada em reticulado. Criptografia polinomial multivariada.

ABSTRACT

A study of post-quantum cryptography algorithms and its costs is presented through the monitoring of the standardization process that NIST is carrying out. Hence, it was possible to carry out a study of some algorithms that are in the standardization phase and that already passed the first elemination stages. This study aimed to understand how these algorithms work and its costs. The NIST process is not over yet, but we can already see improvements in some algorithms, where the creators managed to increace efficiency as well as decrease the size of keys and signature.

Keywords: Cryptography. Post-Quantum Cryptography. Code-based Cryptography. Hash-based Cryptography. Lattice-based Cryptography. Multivariate polynomial cryptography.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

FTS	Few Time Signature
HORS	Hash to Obtain Random Subset
HORST	HORS Tree
OTS	One Time Signature
WOTS	Winternitz One Time Signature

SUMÁRIO

1 INTRODUÇÃO

Computadores quânticos são máquinas que exploram fenômenos da mecânica quântica (BERNSTEIN; BUCHMANN; DAHMEN, 2009; NIST, 2016). Enquanto um computador clássico está em um estado fixo como uma string de bits que representa seu conteúdo em memória, o estado de um computador quântico pode ser uma superposição de vários estados (BALAMURUGAN *et al.*, 2021). Essa característica dá aos computadores quânticos a possibilidade de computar mais ao mesmo tempo (paralelismo real). Desta forma, a computação quântica constitui um novo paradigma de computação em que se espera resolver problemas complexos, que necessitam de muito mais poder computacional para a sua solução do que, hoje, os computadores clássicos conseguem dispor (BALAMURUGAN *et al.*, 2021).

Em 1994, Peter Shor desenvolveu um algoritmo quântico para fatoração de números primos e logaritmos e mostrou que tais problemas podem ser resolvidos eficientemente por computadores quânticos (SHOR, 1994). Assim, com o desenvolvimento de um computador quântico funcional, todos os sistemas de criptografia de chave pública se tornam vulneráveis (CHEN, L. *et al.*, 2016; BERNSTEIN; BUCHMANN; DAHMEN, 2009). Essa constatação impulsionou as pesquisas tanto no desenvolvimento de computadores quânticos quanto no desenvolvimento de novos algoritmos criptográficos que sejam resistentes à computação quântica.

O argumento de que estados quânticos são frágeis e sujeitos ao acúmulo de erro e de que isto inviabilizaria uma computação em larga escala começou a ser desmontado no final da década de 90, com o desenvolvimento de códigos de correção de erros quânticos e teoremas de limiar (CHEN, L. *et al.*, 2016). Estes teoremas mostram que o erro por operação lógica (porta quântica) em um computador quântico pode ser corrigido de forma confiável e de forma tolerante a falhas. Experimentalistas tem gradualmente desenvolvido hardwares melhores com taxas de erros cada vez menores por porta quântica e acredita-se que, em aproximadamente 20 anos, um computador suficientemente poderoso computacionalmente será construído e poderá quebrar essencialmente todos os esquemas de chave pública atualmente em uso (CHEN, L. *et al.*, 2016). Em particular, os algoritmos RSA, DSA e sistemas de criptografia de curvas elípticas serão afetados.

Visando assegurar a segurança da informação, a comunidade de pesquisa em sistemas criptográficos e segurança voltou esforços para a implementação da chamada *c*riptografia pós-quântica, que contempla um conjunto de técnicas de criptografia que não sejam vulneráveis aos ataques realizados por computadores quânticos (PERELMUTER, 2019). A ideia da criptografia pós-quântica é encontrar novas primitivas para a critptografia que viabilizem a manutenção da infraestrutura de chaves públicas (CHEN, L. *et al.*, 2016). Neste sentido, o NIST lançou um programa de padronização de

criptografia pós-quântica com o propósito de solicitar, avaliar e padronizar um ou mais algoritmos de criptografia de chave pública (DRAGOI *et al.*, 2018). A primeira rodada se iniciou em 20 de dezembro de 2016, onde houve a chamada formal para propostas e o limite para envio foi 30 de novembro de 2017. Os algoritmos que entraram nesta primeira rodada foram divulgados no dia 21 de dezembro de 2017. Os candidatos que foram classificados para a segunda rodada foram anunciados em 30 de janeiro de 2019. A terceira rodada aconteceu no dia 22 de julho de 2020 (NIST, 2020b).

Neste trabalho, acompanhou-se o processo de padronização do NIST no que se refere a algoritmos criptográficos pós-quânticos, estudando esta família de técnicas e suas garantias em termos de segurança. Nem todas as propostas de melhorias dos algoritmos são aqui apresentadas, pois seria inviável visto que estes algoritmos ainda estão em processo de padronização. Os principais objetivos deste trabalho são, portanto, apresentar ao leitor detalhes dos algoritmos em sua versão inicial para que seja mais fácil compreender as melhorias propostas, servindo como base para o entendimento do tema e também, realizar um estudo dos custos dos algoritmos pós-quânticos ora propostos.

No Capítulo 2, fala-se sobre algoritmos de criptografia assimétrica, focando em algoritmos para criptografia pós-quântica, famílias e algoritmos propostos para a padronização. No Capítulo 3, apresenta-se uma comparação entre as técnicas, onde são mostrados os tamanhos de chaves e assinaturas e os pontos fortes e fracos de cada algoritmo, visando uma comparação entre eles. O Capítulo 4, apresenta as considerações finais deste trabalho.

2 ALGORITMOS DE CRIPTOGRAFIA ASSIMÉTRICA

Este capítulo apresenta os diversos esquemas de criptografia assimétrica presentes na literatura. Esses esquemas são usados para assinatura digital, criptografia de dados e geração de chaves assimétricas. Inicialmente, apresentamos o algoritmo RSA, amplamente usado na internet hoje, e diversos esquemas propostos para substituí-lo na era pós-quântica. Apresentamos ainda as quatro famílias de algoritmos pós-quânticos e seus principais esquemas criptográficos segundo o NIST (CHEN, L. *et al.*, 2016).

2.1 *RIVEST-SHAMIR-ADLEMAN* (RSA)

A segurança do algoritmo RSA, amplamente utilizado para criptografar e descriptografar conteúdos transmitidos digitalmente, está baseada em um problema matemático que, até hoje, não possui solução eficiente: a fatoração de números primos muito grandes. O problema consiste em descobrir quais são os números primos que multiplicados uns pelos outros, resultam no número original - o que é utilizado como chave para criptografar e assinar mensagens. Para ilustrar a utilidade do RSA, tomemos Alice e Bob como usuários e uma mensagem m. Combinando-se as chaves públicas e privadas dos usuários Alice e Bob, é possível garantir que, se Alice enviar uma mensagem m assinada com sua chave privada e cifrada com a chave pública de Bob, apenas Bob vai conseguir ler a mensagem m de Alice, e ainda que Bob poderá ter certeza que Alice é a autora da mensagem original (PERELMUTER, 2019).

Até agora, apesar de esforços de estudiosos ao redor do mundo, não foi encontrado um algoritmo que consiga fatorar um número grande em fatores primos em um intervalo de tempo aceitável. Por exemplo, a fatoração de um número de 232 dígitos, representado por 768 bits, foi conseguida em 2009 após um conjunto de mais de cem computadores trabalhar no problema por dois anos - e quanto maior o número de dígitos, muito maior será a demora. A questão crítica é que até hoje não foi provado matematicamente que esse problema é impossível de ser resolvido - em outras palavas, ainda não é possível determinar se de fato não existe uma forma eficiente de fatorar um número em seus fatores primos (PERELMUTER, 2019).

O esquema RSA é uma cifra de bloco em que o texto claro e o cifrado são inteiros entre 0 e $n-1$, para algum n. Um tamanho típico para n é 1024 bits, ou 309 dígitos decimais. Ou seja, n é menor que 2^{1024}. O texto claro é criptografado em blocos, com cada um tendo um valor binário menor que algum número n. Ou seja, o tamanho do bloco precisa ser menor ou igual a $\log_2(n)+1$ (STALLINGS, 2015). Uma versão didática do algoritmo é descrito por W. Stallings em (STALLINGS, 2015), onde a encriptação e decriptação, para algum bloco de texto claro M e bloco de texto cifrado C, tem a seguinte forma:

$$C = M^e \mod n \tag{1}$$

$$M = C^d \mod n = (M^e)^d \mod n = M^{ed} \mod n \tag{2}$$

onde e e d são números escolhidos conforme descrito abaixo.

Tanto o emissor quanto o receptor precisam conhecer o valor de n. O emissor conhece o valor de e, e somente o receptor sabe o valor de d. Assim, esse é um algoritmo de encriptação de chave pública com uma chave pública $PU = e, n$ e uma chave privada $PR = d, n$. Para que esse algoritmo seja satisfatório, à encriptação de chave pública, Stallings lista alguns requisitos que precisam ser atendidos (STALLINGS, 2015):

1. É possível encontrar valores de e, d, e n tais que $M^{ed} \mod n = M$ para todo $M < n$.
2. É relativamente fácil calcular $M^e \mod n$ para todos os valores de $M < n$.
3. Conhecendo e e n é inviável determinar d.

Precisamos encontrar um relacionamento na forma:

$$M^{ed} \mod n = M \tag{3}$$

Esse relacionamento se mantém se e e d forem inversos multiplicativos módulo $\phi(n)$, onde $\phi(n)$ é a função totiente ou função $\phi()$ de Euler. A função totiente de Euler é expressa da seguinte forma:

$$\phi(pq) = (p-1)(q-1) \tag{4}$$

e o relacionamento entre e e d pode ser expresso como:

$$ed \mod \phi(n) = 1 \tag{5}$$

que é equivalente a dizer:

$$ed \equiv 1 \mod \phi(n) \tag{6}$$

que podemos transformar em:

$$d \equiv e^{-1} \mod \phi(n) \tag{7}$$

e e d são inversos multiplicativos $\mod \phi(n)$.

O algoritmo funciona da seguinte forma. Escolhemos dois números primos p e q que são privados e escolhidos, calculamos n

$$n = pq \tag{8}$$

Selecionamos e no qual será um valor público e escolhido, tal que e seja relativamente primo de $\phi(n)$ através de:

$$mdc(\phi(n),\ e) = 1;\ tal\ que\ 1 < e < \phi(n) \tag{9}$$

Determinamos d da seguinte forma:

$$d \equiv e^{-1}(\mod \phi(n)) \tag{10}$$

A chave privada consiste em $\{d, n\}$ e a chave pública em $\{e, n\}$. Suponha que Alice tenha publicado sua chave pública e que o usuário Bob queira enviar uma mensagem M para Alice. Então, Bob calcula $C = M^e \mod n$ e transmite C. Ao receber esse texto cifrado, Alice descriptografa calculando $M = C^d \mod n$.

2.2 CRIPTOGRAFIA PÓS QUÂNTICA

O objetivo da criptografia pós-quântica é desenvolver sistemas criptográficos que sejam seguros contra computadores clássicos e quânticos, e possam interoperar com protocolos de comunicação e redes existentes hoje (CHEN, L. *et al.*, 2016). Desta forma, os mesmos aparelhos eletrônicos (ex. celulares) continuarão funcionando de maneira segura, bastando apenas trocar o algoritmo criptográfico usado.

A ideia de criptografia pós-quântica foi primeiramente proposta nos anos de 1970 por Stephen Wiesner e Charles H. Bennet da IBM e Gilles Brassard da Universidade de Montreal (GISIN *et al.*, 2002). Esta, desafia a linha divisória de problemas tratáveis e intratáveis.

A busca por algoritmos considerados resistentes a ataques de computadores clássicos e quânticos focou em algoritmos de infraestrutura de chave pública pois a construção de um computador quântico de grande escala tornaria os atuais esquemas de chaves públicas inseguros. Em particular, isso inclui aqueles baseados na dificuldade de fatoração de inteiros, como RSA apresentado na seção anterior, como também aqueles baseados na dificuldade do problema do logaritmo discreto (CHEN, L. *et al.*, 2016).

O que é pretendido na padronização do NIST é explicitado em (NIST, 2017a). Os algoritmos submetidos para a primeira rodada do NIST podem ser encontrados em (NIST, 2017b). Também pode-se encontrar em (NIST, 2017c) os algoritmos da segunda rodada, bem como os da terceira rodada em (NIST, 2017d). Estas referências trazem além dos algoritmos, os seus websites, autores e comentários.

Segundo Sendrier (SENDRIER, 2017), as principais técnicas de criptografia pós-quânticas disponíveis são: criptografia baseada em código (*Code-based Cryptography*), criptografia baseada em hash (*Hash-based Cryptography*), criptografia baseada em reticulados (*Lattice-based Cryptography*) e criptografia polinomial multivariável (*Multivariate Polynomial Cryptography*).

Uma das mais antigas proposições de algoritmos criptográficos resistentes a computadores quânticos faz referência a *McEliece* em 1978 (MCELLICE, 1978), que propôs um sistema de criptografia de chave pública baseada em teoria de código. Tal proposição se beneficia de um algoritmo realmente eficiente como também possui uma base matemática forte (DRAGOI *et al.*, 2018). Desde a contribuição de McEliece, outros sistemas baseados em correção de erros foram propostos. Embora os algoritmos sejam rápidos, a maior parte dos algoritmos baseados em código sofrem por terem tamanho de chaves grande (CHEN, L. *et al.*, 2016).

Assinaturas baseadas em hash são assinaturas digitais construídas usando funções de hash. Sua segurança, contra ataques quânticos, é bem entendida. Muitos dos mais eficientes esquemas de assinaturas baseados em hash tem a desvantagem de que o assinante deve manter um registro do número exato das mensagens assinadas anteriormente, e qualquer erro neste registro irá resultar em insegurança. Assim, pode-se produzir somente um número limitado de assinaturas (CHEN, L. *et al.*, 2016).

Construções criptográficas baseadas em reticulados são baseadas na dificuldade de resolver problemas envolvendo reticulados. O problema mais básico envolvendo esta técnica é o problema do caminho mínimo (BERNSTEIN; BUCHMANN; DAHMEN, 2009). Para Bernstein et al. (BERNSTEIN; BUCHMANN; DAHMEN, 2009), criptografia baseada em reticulados mantém uma grande promessa para a criptografia pós-quântica pois tais problemas desfrutam de provas de segurança muito fortes, implementações relativamente eficientes, bem como grande simplicidade.

Criptografia polinomial multivariada é o estudo de sistemas de criptografia de chave pública onde a função de sentido único toma a forma de um mapa polinomial quadrático sobre um corpo finito (BERNSTEIN; BUCHMANN; DAHMEN, 2009).

Acreditando que as técnicas supracitadas resistem a computadores clássicos e quânticos (BERNSTEIN; BUCHMANN; DAHMEN, 2009), temos os algoritmos que entraram na terceira rodada do NIST listados nos quadros a seguir. Nos Quadros 1 e 2 temos os 7 algoritmos finalistas, que avançaram para a terceira rodada e nos Quadros 3 e 4 temos os 8 algoritmos que não foram finalistas, mas que porém, também avançaram para a terceira rodada.

Os algoritmos finalistas continuarão a ser revisados e considerados para a padronização na conclusão da terceira rodada. Os candidatos alternativos que estão na terceira rodada, potencialmente ainda podem ser padronizados, embora provavelmente isso não ocorra no final da terceira rodada. O NIST espera ter uma quarta rodada de avaliação para os candidatos alternativos. Vários destes algoritmos tem pior desempenho que os finalistas, porém, podem ser selecionados para padronização baseados na alta confiança em sua segurança. Já outros candidatos possuem eficiência aceitável mas requerem análise adicional (NIST, 2020a).

Quadro 1 – Criptografia de Chave Pública e Algoritmos de Estabelecimento de Chaves.

Algoritmo	**Família**
McEliece Clássico	Criptografia Baseada em Código
CRYSTALS-KYBER	Criptografia Baseada em Reticulados
NTRU	Criptografia Baseada em Reticulados
SABER	Criptografia Baseada em Reticulados

Quadro 2 – Algoritmos de Assinatura Digital.

Algoritmo	**Família**
CRYSTALS-DILITHIUM	Criptografia Baseada em Reticulados
FALCON	Criptografia Baseada em Reticulados
Rainbow	Criptografia Polinomial Multivariada

Quadro 3 – Cadidatos Alternativos para Criptografia de Chave Pública e Algoritmos de Estabelecimento de Chaves.

Algoritmo	**Família**
BIKE	Criptografia Baseada em Código
FrodoKEM	Criptografia Baseada em Reticulados
HQC	Criptografia Baseada em Código
NTRU Prime	Criptografia Baseada em Reticulados
SIKE	Criptografia Baseada em Reticulados

Quadro 4 – Candidatos Alternativos para Algoritmos de Assinatura Digital.

Algoritmo	**Família**
GeMSS	Criptografia Polinomial Multivariada
Picnic	Criptografia Baseada em Hash
SPHINCS+	Criptografia Baseada em Hash

2.2.1 Categorias de Segurança do NIST

Em (NIST, 2016) se fala sobre incertezas na estimativa de segurança dos algoritmos criptográficos pós-quânticos. Essas incertezas advêm da possibilidade de novos algoritmos quânticos serem descobertos levando a novos ataques criptográficos como

também da habilidade limitada de prever as características de desempenho dos futuros computadores quânticos, seus custos, velocidade e tamanho de memória.

Ao invés de definir a força dos algoritmos submetidos utilizando o número de "bits de segurança", o NIST define uma coleção de categorias de segurança. Cada categoria definida por uma referência comparativamente fácil de analisar, cuja segurança servirá como base para uma ampla variedade de métricas que o NIST considera potencialmente relevantes para a segurança. Um dado sistema criptográfico pode ser instanciado usando conjuntos diferentes de parâmetros com objetivo de se encaixar em categorias distintas. O objetivo desta classificação é:

- Facilitar comparações de desempenho significativas entre os algoritmos submetidos, garantindo, na medida do possível, que os conjuntos de parâmetros comparados possuam segurança comparável.
- Permitir que o NIST tome decisões futuras prudentes sobre quando fazer a transição para chaves mais longas.
- Ajudar os remetentes dos algoritmos a fazerem escolhas consistentes e sensíveis quando se trata de primitivas simétricas para usar em macanismos de "padding" ou outros componentes de seus esquemas que requerem criptografia simétrica.
- Entender melhor a troca de segurança e desempenho envolvidos em um determinado algoritmo.

Em (NIST, 2016), afirma-se que o NIST irá basear sua classificação em um intervalo de força de segurança oferecidos pelos padrões existentes de criptografia simétrica do NIST, onde espera-se que o algoritmo ofereça resistência à criptoanálise quântica. O NIST definire uma categoria separada para cada um dos requisitos de segurança, ordenados em ordem crescente de segurança:

1 Qualquer ataque que quebre uma relevante parte da definição de segurança deve exigir recursos computacionais comparáveis ou superiores àqueles necessários para uma busca de chave em uma cifra de bloco com uma chave de 128 bits (por exemplo, AES128).

2 Qualquer ataque que quebre uma relevante parte da definição de segurança deve exigir recursos computacionais comparáveis ou superiores àqueles necessários para uma busca de colisão em uma função de hash de 256 bits (por exemplo, SHA256/SHA3-256).

3 Qualquer ataque que quebre uma relevante parte da definição de segurança deve exigir recursos computacionais comparáveis ou superiores àqueles necessários para uma busca de chave em uma cifra de bloco com uma chave de 192 bits (por exemplo, AES192).

4 Qualquer ataque que quebre uma relevante parte da definição de segurança deve exigir recursos computacionais comparáveis ou superiores àqueles necessários para uma busca de colisão em uma função de hash de 384 bits (por exemplo, SHA384/SHA3-384).

5 Qualquer ataque que quebre uma relevante parte da definição de segurança deve exigir recursos computacionais comparáveis ou superiores àqueles necessários para uma busca de chave em uma cifra de bloco com uma chave de 256 bits (por exemplo, AES256).

2.2.2 Criptografia baseada em código

A Teoria de Código se originou com o advento dos computadores (CHEROWITZO, 2020a). Enquanto Richard Hamming estava trabalhando nos Laboratórios Bell, lhe ocorreu a possibilidade da máquina ser capaz de saber que ocorreu um erro e com base nisso também na possibilidade da máquina corrigir o erro. Com base nisso, Hamming planejou uma maneira de codificar a informação de tal forma que se um erro fosse detectado, também poderia ser corrigido. Baseado nisso, Claude Shannon desenvolveu a estrutura teórica para a ciência da teoria da codificação (CHEROWITZO, 2020a).

Criptografia baseada em código é uma das técnicas matemáticas que possibilita a construção de sistemas de criptografia de chave pública considerados seguros contra um adversário equipado com um computador quântico. Porém, este sistema não possui um esquema de assinatura digital satisfatório (SENDRIER, 2017), o que não invalida a técnica para criptografia de dados.

Neste esquema, a primitiva algorítmica usa um código de correção de erro C. Esta primitiva pode consistir em adicionar um erro para uma palavra de C ou na computação de uma síndrome em relação a uma matriz de verificação de paridade de C (BERNSTEIN; BUCHMANN; DAHMEN, 2009).

A maioria das primitivas de criptografia baseadas em código sofre por ter tamanho de chaves muito grande. Novas variantes tem introduzido mais estruturas nos códigos em uma tentativa de reduzir o tamanho das chaves, entretanto, a estrutura adicionada também tem conduzido a ataques com sucesso em algumas propostas (CHEN, L. *et al.*, 2016).

A ideia central de teoria de códigos se dá por meio de um par específico de modelos matemáticos, a Fonte Simétrica Binária e o Canal Simétrico Binário (MCELIECE, 2002). Fonte Simétrica Binária é um objeto que emite uma ou duas possibilidades de símbolos, que consideramos ser “0” e “1” a uma taxa de R símbolos por unidade de tempo. Estes bits emitidos pela fonte são aleatórios, e um “0” é tão provável de ser emitido quanto “1”. Assumimos que a taxa da fonte R é continuamente variável e que pode assumir qualquer valor não negativo. Já o Canal Simétrico Binário é um objeto

através do qual é possível transmitir um bit por unidade de tempo. Entretanto, o canal não é completamente confiável: há uma probabilidade p (0 a 50%) de que o bit de saída não seja o mesmo que o de entrada.

2.2.2.1 Esquema de McEliece

O esquema de McEliece é um dos candidatos para uma padronização de criptografia de chave pública pós-quântica (SENDRIER, 2017). Este esquema participou da primeira rodada de algoritmos para padronização do NIST e foi classificado para a segunda rodada. Em 22 de Julho de 2020, foi anunciado sua classificação para a terceira rodada (NIST, 2019).

Robert McEliece propôs o primeiro sistema de criptografia baseado em código em 1978, onde sua ideia original foi usar como texto cifrado uma palavra de um código de correção de erro cuidadosamente escolhido usando um código Goppa binário ao qual erros aleatórios foram adicionados. Uma base arbitrária de um código, onde usa-se uma matriz geradora, é a chave pública, permitindo qualquer um encriptar.

Segundo Sendrier (SENDRIER, 2017), a segurança do esquema de McEliece depende de duas suposições computacionais: a decodificação genérica é difícil na média e a chave pública é difícil de distinguir de uma matriz aleatória. Sendrier afirma ainda que o problema é intratável e, portanto, o sistema é seguro contra qualquer adversário computacionalmente limitado. Num comparativo apresentado em (SENDRIER, 2017), mostra-se que para atingir uma segurança "clássica" de 128 bits com o esquema de McEliece original usando códigos Goppa binários, os códigos devem ser provavelmente de dimensão $k = 3.376$ e tamanho $n = 4.096$ corrigindo $t = 60$ erros. O tamanho da chave pública é 303.840 bytes, e a expansão entre o texto claro e o texto cifrado é de aproximadamente 20%. Para a mesma segurança contra adversários quânticos, o tamanho de bloco deve aumentar por um fator de 2, e o tamanho de chave por um fator de 4. Sendrier aponta as seguintes vantagens e desvantagens do esquema de McEliece e suas variantes O esquema de McEliece é um dos candidatos para uma padronização de criptografia de chave pública pós-quântica (SENDRIER, 2017).
Vantagens:

- A segurança é bem entendida.
- Resistiu por mais de 40 anos de pesquisas minuciosas.
- A encriptação e decriptação é computacionalmente eficiente.

Desvantagens:

- Não há esquema de assinatura prático.
- A chave pública é grande, na ordem de 1MB para segurança de longo prazo.

- A suposição de indistinguibilidade do código deve ser mais explorada.

Antes de iniciar uma descrição sobre o funcionamento do algoritmo, Singh em (SINGH, 2019) nos apresenta a noção de um código linear e uma matriz geradora. Sejam k e n inteiros positivos, $k \leq n$. Um código linear é um espaço de dimensão k de subespaço de $(\mathbb{Z}_2)^n$, o vetor espaço de todas as n tuplas binárias. Um código linear $[n, k]$ C, é um subespaço de dimensão k de $(\mathbb{Z}_2)^n$. Uma matriz geradora para um código linear $[n, k]$ C, é uma matriz binária $k \times n$ na qual as linhas formam a base para C.

Singh (SINGH, 2019) também dá a definição da distância de um código linear. Sejam x,$y \in (\mathbb{Z}_2)^n$, onde $x = (x_1, ..., x_n)$ e $y = (y_1, ..., y_n)$. A distância de Hamming é o número de coordenadas no qual x difere de y e está definida na Equação 11. Seja C um código $[n, k]$-linear, a distância de C está definida na Equação 12. Um código $[n, k, d]$ linear é um código $[n, k]$ linear C, onde $dist\hat{a}ncia(C) \geq d$.

$$dist\hat{a}ncia(x, y) = |\{i : 1 \leq i \leq n, x_i \neq y_i\}| \tag{11}$$

$$dist\hat{a}ncia(C) = \min\{dist(x, y) : x, y \in C, x \neq y\}. \tag{12}$$

e apresenta a definição de complemento ortogonal e matriz de checagem de paridade como a seguir. Sendo dois vetores x,$y \in (\mathbb{Z}_2)^n$, onde $x = (x_1, ..., x_n)$ e $y = (y_1, ..., y_n)$, são ortogonais se:

$$\sum_{i=1}^{n} x_i y_i \equiv 0 (mod\ 2).$$

O complemento ortogonal de um código $[n, k, d]$ linear, C, consiste de todos os vetores que são ortogonais para todos os valores em C. Este conjunto de vetores é denotado por $C^{\perp}$ e é chamado de código duplo de C. A matriz de paridade para um código $[n, k, d]$ linear C tendo a matriz geradora G é uma matriz geradora H para $C^{\perp}$. Esta matriz H é uma matriz $(n - k)$ por n.

Em meio a este processo, precisamos de uma forma de corrigir os erros que ocorreram através de uma transmissão, (SINGH, 2019) nos apresenta uma descrição desse processo. O propósito de um código corretor de erro é corrigir erros aleatórios que ocorrem na transmissão de um dado binário através de um canal com ruído. Isso é feito da seguinte maneira: Seja G uma matriz geradora para um código $[n, k, d]$ linear. Suponha que x é o binário da k-tupla que desejamos transmitir. Então, codificamos x como uma n-tupla $y = xG$ e nós transmitimos y através do canal.

Agora, vamos supor que Bob recebe a n-tupla r, no qual pode não ser a mesma que y. Ele irá decodificar r usando a estratégia da decodificação do vizinho mais

próximo. A ideia é que Bob encontra uma palavra de código $y' \neq r$ que tem a mínima distância para r. Tal palavra de código será chamada de vizinho próximo a r e será denotado por $vp(n)$. O processo de computar $vp(n)$ é chamado de decodificação de vizinho mais próximo.

Após decodificar r para $y' = vp(n)$, Bob iria determinar a k-tupla x' tal que $y' = x'G$. Bob está esperando que $y' = y$, então $x' = x$.

A decodificação do vizinho mais próximo seria feita na prática. O número de possíveis palavras de código é igual a 2^k. Se Bob comparar r para cada palavra de código, então ele terá que examinar 2^k vetores, no qual é um número exponencialmente grande comparado a k (SINGH, 2019). O algoritmo de decodificação óbvio não é um algoritmo de tempo polinomial. Para cada polinômio irredutível de grau t sobre $GF(2^m)$, existe um código Goppa binário irredutível de tamanho $n = 2^m$, dimensão $k \geq n - tm$, capaz de corrigir qualquer padrão de t ou menos erros. Além disso, existe um algoritmo rápido (operando em O(nt)) para decodificação destes códigos (MCELLICE, 1978).

A chave pública é uma matriz calculada como $G_{publica} = SGP$, onde G é uma matriz geradora de código Goppa, P uma matriz de permutação e uma matriz não singular S randomicamente escolhida como máscara da estrutura de G (SAMOKHINA; TRUSHINA, 2017).

O esquema de criptografia de chave pública de McEliece é composto por três algoritmos: Geração de Chave, Ciframento, Deciframento. O algoritmo de geração de chave recebe como entrada os inteiros n, m, k, t, q tal que $k < n$ e $t < n$ e retorna um par de chaves pública e privada (pk, sk) (BUCERZAN; VLAD; TALÉ KALACHI, 2017). O inteiro n representa o tamanho do código, o k representa a dimensão do código sobre um campo $\mathbb{F}_q$ e o t representa o grau do polinômio Goppa. Os valores de n, k, e t são parâmetros disponíveis publicamente (SINGH, 2019).

Para o algoritmo de Geração de Chaves temos que $GeraçãoDeChave(n, m, k, t, q) = (pk, sk)$, onde pk é a chave pública e sk é a chave privada (BUCERZAN; VLAD; TALÉ KALACHI, 2017). Em (MCELLICE, 1978) é descrito que devemos escolher um valor desejável para n e t e randomicamente, selecionamos um polinômio irredutível de grau t sobre $GF(2^m)$. Em seguida, produzimos uma matriz geradora $G_{n\times k}$ para o código, no qual pode ser em formato canônico, por exemplo, escala reduzida de linha. Após gerar G, o embaralhamos selecionando uma matriz singular densa aleatória $S_{k\times k}$, e uma matriz de permutação aleatória $P_{n\times n}$. Então, calculamos $G_{publica} = SGP$, no qual gera um código linear com a mesma taxa e distância mínima que o código gerado por G. Nós chamamos $G_{publica}$ a matriz geradora pública, uma vez que será divulgada ao mundo externo.

Assumindo que o código escolhido pode corrigir até t erros, Samokhina e Trushina (SAMOKHINA; TRUSHINA, 2017) descrevem que a mensagem m pode ser mascarada por um vetor z gerado aleatoriamente com o peso Hamming não excedendo t. O cifrador é calculado como $z = mG_{publica} + e$. O peso Hamming de uma palavra é

determinado por seus números diferentes de zero (SENDRIER, 2017). Neste caso, o vetor e não pode exceder t elementos diferentes de zero.

Para Alice gerar um par de chaves pública e privada, são realizados os passos descritos a seguir (BUCERZAN; VLAD; TALÉ KALACHI, 2017; SINGH, 2019):

1. Alice seleciona um código-[n, k] Goppa binário, onde $k \times n$ é o tamanho da matriz geradora G, capaz de corrigir até t erros.
2. Então, ela seleciona uma matriz $S_{k\times k}$ aleatória binária não singular e uma matriz de permutação $P_{n\times n}$.
3. Ela calcula $G_{publica} = SGP$.
4. Ela publica sua chave pública $pk = (G_{publica}, t)$.
5. Ela mantém sua chave privada $sk = (S, G, P)$.

Em (SINGH, 2019) e (BUCERZAN; VLAD; TALÉ KALACHI, 2017) são descritos os passos que Bob seguirá caso tenha que mandar uma mensagem para Alice, onde Bob utilizará o algoritmo *Cifragem* dando como parâmetros uma mensagem m e a chave pública de Alice (pk), da forma que $Cifragem(m, pk) = z$.

1. Bob possui uma mensagem de texto binária m de tamanho k.
2. Ele carrega a chave pública $(G_{publica}, t)$ de Alice.
3. Ele gera um vetor aleatório e de n bits com peso Hamming t.
4. Bob calcula o texto cifrado $z = mG_{publica} + e$ e envia para Alice

O processo de deciframento inclui o cálculo de zP^{-1} como um primeiro estágio e o uso de um algoritmo rápido de deciframento para obter m^* como segundo estágio. O cálculo final resulta na obtenção do texto claro multiplicando m^* pela matriz S^{-1} (SAMOKHINA; TRUSHINA, 2017). Bucerzan et al. (BUCERZAN; VLAD; TALÉ KALACHI, 2017) e Singh (SINGH, 2019) apresentam o algoritmo para decifragem, onde $Decifragem(z, pk) = m$. Supondo que Alice tenha recebido o texto cifrado z, ela descriptografa o texto cifrado recebido como:

1. Alice computa P^{-1} usando sua chave privada.
2. Ela computa $zP^{-1} = mSG + eP^{-1}$
3. Finalmente, ela usa o algoritmo de decodificação de códigos Goppa para o código secreto Goppa para determinar o valor de m.

(SINGH, 2019) afirma que há uma série de algoritmos para decodificar códigos Goppa. (BUCERZAN; VLAD; TALÉ KALACHI, 2017) afirma que podemos verificar a corretude do esquema checando:

$$Decifragem(Cifragem(m, pk), sk) = m$$

Um exemplo do esquema de McEliece é apresentado em (STINSON, 2019). A matriz descrita na Equação 13 é a matriz geradora para um código-[7, 4, 3] linear, conhecido como um código Hamming.

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{13}$$

Supondo que Bob escolha as matrizes das Equações 14 e 15, então a matriz geradora pública consiste da Equação 16.

$$S = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix} \tag{14}$$

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \tag{15}$$

$$G_{publica} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix} \tag{16}$$

Suponha que Alice encripta o texto plano $m = (1, 1, 0, 1)$ usando o vetor $e = (0, 0, 0, 0, 1, 0, 0)$ com um vetor aleatório de peso 1. O texto cifrado é computado da seguinte forma:

$$
\begin{aligned}
z &= mG_{publica} + e \\
&= (1,1,0,1)\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix} + (0,0,0,0,1,0,0) \\
&= (0,1,1,0,0,1,0) + (0,0,0,0,1,0,0) \\
&= (0,1,1,0,1,1,0).
\end{aligned}
$$

Quando Bob recebe o texto cifrado z, ele calcula:

$$
\begin{aligned}
z^* &= zP^{-1} \\
&= (0,1,1,0,1,1,0)\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \\
&= (1,0,0,0,1,1,1).
\end{aligned}
$$

Em (MANFERDELLI, 2020), Manferdelli mostra como fazer a decodificação Hamming. Por definição, uma matriz de um código $[7,4]$ Hamming possui uma matriz de checagem de paridade H, onde:

$$
H = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}
$$

Sendo assim, temos que H^T é:

$$
H^T = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}
$$

Fazendo a multiplicação de z^* por H^T temos que:

$$z^* \times H^T = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$$

Podemos observar que a multiplicação de z^* por H^T resultou em (001). Com base nisso, procuramos na matriz H^T a linha que coincide com este resultado. Observando a matriz transposta, podemos notar que a linha 7 coincide com o resultado. Então, há um erro no sétimo bit de z^*. Como o sétimo bit é 1 e já descobrimos que ele está errado, então o bit só pode ser 0.

Em (CHEROWITZO, 2020b), Cherowitzo afirma que após Bob obter z^*, ele decodifica z^* através de um algoritmo rápido, sendo que a decodificação Hamming é utilizada neste exemplo. Ainda diz que z^* é $(1,1,1,0)^T$, então o erro ocorre na posição 7. Então, Bob agora tem a palavra de código $m^* = (1,0,0,0,1,1,0)$. Por causa da escolha esperta para G, Bob sabe que $m_0^* = (1000)$ e que $mS = m_0^*$ e agora ele pode obter m multiplicando m_0^* pela matriz:

$$S^{-1} = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{aligned} m &= m_0^* S^{-1} \\ &= (1,0,0,0) \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix} \\ &= (1,1,0,1). \end{aligned}$$

Isso é de fato o que a Alice encriptou.

Observando o tamanho das chaves do esquema de McEliece e algumas de suas variações, Singh (SINGH, 2019) descreve a alocação de chaves para o esquema de McEliece, Niederreiter e McEliece Clássico que serão apresentadas a seguir. Relembrando os parâmetros do esquema de McEliece sabemos que a chave pública é uma

matriz $G_{publica(k\times n)}$. Para a chave privada temos a matriz aleatória $S_{k\times k}$, a matriz de permutação $P_{n\times n}, t$ como sendo o grau do polinômio sobre $GF(2^m)$ e n como o tamanho de código Goppa binário.

Os parâmetros fornecidos na construção original foram:

$$\begin{aligned} n &= 1024 = 2^{10} \\ t &= 50 \\ m &= 10 \\ k &= n - mt = 524 \end{aligned}$$

Com isso, o tamanho da chave pública é:

$$kn = 524 \times 1024 = 536676\ bits \approx\ 66KB$$

Para o tamanho da chave privada temos:

$$\begin{aligned} &(k^2 + n^2) + (t \times m) + (n \times m) \\ &= (274576 + 1048576)\ +\ 500\ +\ 10240 \\ &= 1333892\ bits \approx\ 162.8KB \end{aligned}$$

Para o esquema de Niederreiter, são usados os mesmos parâmetros fornecidos no esquema de McEliece resultando em uma chave pública de 62KB e uma chave privada de 159.8KB.

Outra variante que inclusive é candidata a padronização do NIST, é o esquema de McEliece Clássico. As duas variantes propostas são chamadas de mceliece6960119 e mceliece8192128. A variante mceliece6960119 possui chave pública de 1MB e chave privada de 13.6KB. Já a variante mceliece8192128, possui chave pública de 1.3MB e chave privada de 13.75MB.

Quadro 5 – Tamanho de chaves de esquemas baseados em Código.

Esquema	**Chave Pública**	**Chave Privada**
McEliece	66KB	162.8KB
Niederreiter	62KB	159.8KB
mceliece6960119	1MB	13.6KB
mceliece8192128	1.3MB	13.75KB

Em (SAMOKHINA; TRUSHINA, 2017), Samokhina e Trushina afirmam que o algoritmo de McEliece Clássico é relativamente rápido e opera 10 vezes mais rápido que o sistema RSA padrão, porém possui a desvantagem da chave pública ser bastante grande.

2.2.3 Criptografia baseada em *hash*

Esquemas de assinatura baseadas em hash foram criadas por Ralph Merkle. Merkle começou em esquemas de assinatura únicos, em particular aquela de Lamport e Diffie. Esquemas de assinatura únicos requerem apenas uma função de sentido único. A construção inicial de Merkle não foi suficientemente eficiente, particularmente em comparação ao esquema de assinatura RSA. No entanto, nesse meio tempo, muitas melhorias foram propostas. Agora, assinaturas baseadas em hash são a alternativa mais promissora para o RSA e esquemas de assinaturas de curvas elípticas (BERNSTEIN; BUCHMANN; DAHMEN, 2009).

Esquemas de assinaturas baseados em hash podem ser divididos em com e sem estado. Neste contexto, com estado significa que a chave secreta muda após cada assinatura. No caso de uma chave secreta ser utilizada duas vezes, todas as garantias de segurança desaparecem. Na prática, em muitos cenários, manter um estado se torna complicado. Entretanto, esquemas com estado atuais são aqueles considerados para padronização pois estes esquemas são muito mais eficazes em termos de tamanho de assinatura e velocidade de assinatura do que as alternativas sem estado (GROOT BRUINDERINK; HÜLSING, 2018).

Segundo Bernstein et al. (BUCHMANN; DAHMEN; SZYDLO, 2009), esquemas de assinatura baseados em hash são os melhores candidatos para assinatura pós-quântica. Sua segurança até mesmo contra ataques quânticos é bem entendida (CHEN, L. *et al.*, 2016). Esquemas de assinaturas baseadas em hash usam funções de hash criptográficas. Sua segurança depende da resistência à colisão de uma função de hash. A existência de funções de hash resistentes à colisão pode ser vista como uma necessidade mínima para a existência de um esquema de assinatura digital que pode assinar muitos documentos com uma chave privada (BUCHMANN; DAHMEN; SZYDLO, 2009). Apesar de numerosos os esquemas de assinatura que foram construídos desde os anos 1970, somente um pequeno conjunto é usado em larga escala (BUTIN, 2017).

O uso de assinaturas únicas são fundamentais para o funcionamento de esquemas de assinaturas baseadas em hash. A construção de um esquema seguro de assinatura única requer apenas uma função de sentido único. Funções de sentido único são necessárias e suficientes para assinaturas digitais seguras. Esquemas de assinatura única são realmente o mais fundamental tipo de esquemas de assinatura digital. Entretanto, possuem uma desvantagem severa. Um par de chaves consistindo de uma assinatura de chave secreta e uma chave de verificação pública pode somente ser usada para assinar e verificar um único documento. Isto é inadequado para a maioria das aplicações (BUCHMANN; DAHMEN; SZYDLO, 2009). A principal ideia para contornar esse problema foi de Merkle e consiste em utilizar muitos pares de chaves de assinatura única e combiná-los em uma estrutura usando uma árvore de

hash. Uma árvore de hash é uma estrutura de dados hierárquica que repetidamente usa uma função de hash e concatenação para computar nós. Dois argumentos chaves suportam o uso de assinaturas baseadas em hash (BUTIN, 2017).

O primeiro argumento apontado por (BUTIN, 2017) é o da suposição mínima de segurança. Onde esquemas de assinaturas baseadas em hash como o XMSS (Extended Merkle Signature Scheme[1]) requer apenas uma criptografia segura de função de hash criptográfica para ser segura. Está provado que, desde que exista qualquer algoritmo de assinatura seguro, uma instanciação de XMSS existe. Nesse sentido, XMSS e outros esquemas de assinatura exibem requisitos de segurança mínimos.

O segundo argumento apontado por (BUTIN, 2017) é o da natureza genética. Os esquemas de assinaturas baseados em hash podem ser instanciados com qualquer função de hash que atenda critérios simples, resultando em uma flexibilidade tremenda. Estes esquemas podem ser vistos como modelos, com a função de hash subjacente como uma peça que pode ser substituída sem alterar a estrutura geral. Isso é significativo para segurança a longo prazo, porque novas vulnerabilidades de funções de hash podem surgir com o tempo.

Isso mostra que algoritmos de assinatura digitais são de fato funções de hash. Estas funções de hash devem ser resistentes a colisão: se for possível construir dois documentos com a mesma assinatura digital, o esquema de assinatura não pode mais ser considerado seguro. Este argumento mostra que existe esquemas de assinatura baseados em hash enquanto existir qualquer esquema de assinatura digital que pode assinar múltiplos documentos usando uma chave privada (BUCHMANN; DAHMEN; SZYDLO, 2009).

Na sequência apresentamos resumidamente alguns esquemas de assinatura baseados em hash presentes na literatura.

2.2.3.1 One-Time Signature (OTS)

Butin (BUTIN, 2017) afirma que o OTS é o pilar de assinaturas baseadas em hash e podem ser vistas como assinaturas digitais em miniatura. Sua segurança depende inteiramente e exclusivamente da função de hash utilizada.

Um esquema OTS permite usar um par de chaves para assinar uma única mensagem. Se um par de chaves é utilizado para assinar uma segunda mensagem diferente, não há nenhuma garantia de segurança. Sabe-se que se um adversário obtiver controle total sobre as mensagens a serem assinadas, estes esquemas são totalmente quebrados após duas assinaturas. Isto é, a chave secreta pode ser extraída sem esforço. Porém, na prática o OTS com estado é utilizado para assinar o hash de uma mensagem escolhida pelo adversário (GROOT BRUINDERINK; HÜLSING, 2018).

[1] Esquema de Assinatura Estendido de Merkle

2.2.3.1.1 Lamport-Diffie One-Time Signature Scheme (LD-OTS)

O esquema de assinatura única de Lamport-Diffie é descrito em (BUCHMANN; DAHMEN; SZYDLO, 2009). Seja um inteiro positivo n, o parâmetro de segurança do esquema. LD-OTS usa uma função de sentido único:

$$f : \{0,1\}^n \rightarrow \{0,1\}^n \tag{17}$$

e uma função de hash criptográfica:

$$g : \{0,1\}^* \rightarrow \{0,1\}^n \tag{18}$$

Para o processo de geração de chaves, a chave de assinatura X do LD-OTS consiste de $2n$ bits de tamanho n escolhidos uniformemente aleatórios:

$$X = (x_{n-1}[0],\ x_{n-1}[1],\ ...,\ x_1[0],\ x_1[1],\ x_0[0],\ x_0[1]) \in_R \{0,1\}^{(n,\,2n)} \tag{19}$$

A chave de verificação Y do LD-OTS é:

$$Y = (y_{n-1}[0],\ y_{n-1}[1],\ ...,\ y_1[0],\ y_1[1],\ y_0[0],\ y_0[1]) \in \{0,1\}^{(n,\,2n)} \tag{20}$$

onde

$$y_i[j] = f(x_i[j]),\ 0 \leq i \leq n-1,\ j = 0,1 \tag{21}$$

A geração de chaves do LD-OTS requer $2n$ avaliações de f. A assinatura e a verificação de chaves são de $2n$ bits de tamanho n.

Na geração da assinatura do LD-OTS, um documento $M \in \{0,1\}^*$ é assinado com uma chave de assinatura X conforme a Equação (19). Seja $g(M) = d = (d_{n-1}, ..., d_0)$ o hash de M. Então, a assinatura do LD-OTS é:

$$\sigma = (x_{n-1}[d_{n-1}], ..., x_1[d_1], x_0[d_0]) \in \{0,1\}^{(n,n)} \tag{22}$$

A assinatura é uma sequência de n bits, cada um com tamanho n. Eles são escolhidos como uma função da função de hash da mensagem d. O i-ésimo bit nesta assinatura é $x_i[0]$ se o i-ésimo bit em d é 0, caso contrário será $x_i[1]$. Assinar não requer avaliações de f. O tamanho da assinatura é n^2.

Para verificar uma assinatura $\sigma = (\sigma_{n-1}, ..., \sigma_0)$ de M como descrito na Equação (22), o verificador calcula o hash da mensagem $d = (d_{n-1}, ..., d_0)$. Então, checa se

$$(f(\sigma_{n-1}), ..., f(\sigma_0)) = (y_{n-1}[d_{n-1}], ..., y_0[d_0]) \tag{23}$$

A verificação de assinatura requer n avaliações de f.

2.2.3.1.2 Winternitz One-Time Signature (WOTS)

Logo após a publicação do LD-OTS, Robert Winternitz sugere uma melhoria onde reduz o tamanho da mensagem assinada por um fator de cerca de 4 a 8 vezes. O método do Winternitz faz a troca de tempo por espaço, ou seja, o tamanho reduzido é trocado por um aumento de esforço computacional (MERKLE, 1989).

No método de Lamport-Diffie, dado $y = F(x)$ e que y é público e x é privado, um usuário assina um único bit de informação tornando x público ou mantendo-o em segredo (MERKLE, 1989).

No método de Winternitz ainda usamos y e x, fazendo y público e mantendo x privado, mas calculamos y a partir de x aplicando a função F repetidamente (MERKLE, 1989).

Mais detalhes sobre este esquema podem ser encontrados em (MERKLE, 1989) e (BUCHMANN; DAHMEN; SZYDLO, 2009).

2.2.3.1.3 Winternitz One-Time Signature Plus (WOTS+)

O WOTS+ é uma variação do esquema WOTS no qual foi projetado para reduzir o tamanho de assinatura. No esquema WOTS, uma função F é aplicada a uma chave privada várias vezes para produzir uma cadeia de hash. No WOTS+, levamos em conta máscaras de bit. Em cada iteração, antes de aplicar a função de encadeamento F, é realizada uma operação XOR entre a entrada e a máscara de bit que é específica em cada iteração (HÜLSING; RIJNEVELD; SCHWABE, 2016).

O esquema WOTS+ possui alguns parâmetros, sendo n o parâmetro de segurança como também o tamanho da mensagem e da chave privada. w é o parâmetro de Winternitz, onde é um elemento do conjunto $\{4, 16, 256\}$. Estes valores são utilizados para calcular os valores l_1, l_2 e l da seguinte forma (AUMASSON *et al.*, 2020):

(HÜLSING; RIJNEVELD; SCHWABE, 2016) nos apresenta as equações para calcular l_1, l_2 e *l*

$$l_1 = \left\lceil \frac{n}{\log(w)} \right\rceil, l_2 = \left\lfloor \frac{log(l_1(w-1))}{\log(w)} \right\rfloor + 1, l = l_1 + l_2$$

O valor de n determina o comprimento de entrada e saída da função de hash usada para o WOTS+. O valor de n também determina o comprimento das mensagens que podem ser processadas usando o algoritmo de assinatura do WOTS+. Para w, um valor maior resulta em assinaturas menores, porém com operações mais lentas e não sem prejudicar a segurança do esquema. Escolhas para w estão limitadas ao conjunto de valores citados, pois geram uma compensação entre tempo-espaço ótima e de fácil implementação (AUMASSON *et al.*, 2020).

Em (BERNSTEIN; HOPWOOD *et al.*, 2015) a função de encadeamento é definida recursivamente como:

$$c^i(x, r) = F(c^{i-1}(x, r) \oplus r_i),$$

onde temos $i \in \mathbb{N}$ como contador de iteração e os parâmetros $x \in \{0, 1\}^n$ e as máscaras de bit $r = (r_1, ..., r_j) \in \{0, 1\}^{n \times j}$ com $j \geq i$. Em caso i = 0, c retorna x, isto é, $c^0(x, r) = x$. Ou seja, em cada rodada, a função F recebe como parâmetro o resultado de uma operação *xor* bit a bit entre o valor anterior $c^{i-1}(x, r)$ e a máscara de bit r_i. Mais detalhes sobre este algoritmo podem ser encontrados em (HÜLSING, 2017).

2.2.3.2 Few-Time Signature (FTS)

Segundo (SUHAIL *et al.*, 2020), manter o controle do último par de chaves OTS utilizado é considerado uma das maiores desvantagens de esquemas com estado. Para tratar este problema, esquemas sem estado são introduzidos, fazendo com que não seja necessário manter a chave privada não repetida atualizada em cada processo de geração de assinatura. São exemplos de FTS o HORS e HORST, disponíveis nas seções 2.2.3.2.1 e 2.2.3.2.2.

O FTS é uma alternativa ao OTS para utilizar como blocos de construção para esquemas de assinaturas baseadas em hash. Estes esquemas são usados para assinar um número limitado de mensagens para um dado par de chaves (BUTIN, 2017).

2.2.3.2.1 Hash to Obtain Random Subset (HORS)

Para hashs de mensagens de tamanho m, HORS usa dois parâmetros $t = 2^\tau$ para $\tau \in \mathbb{N}$ e $k \in \mathbb{N}$ tal que $m = k \log t = k\tau$. Utiliza uma chave privada consistindo de t valores aleatórios. A chave pública consiste de t hashs destes valores. Uma assinatura consiste de k-elementos de chaves privadas, com índices selecionados como funções públicas da mensagem sendo assinada (BERNSTEIN; HOPWOOD *et al.*, 2015).

2.2.3.2.2 Hash to Obtain Random Subset with Trees (HORST)

O HORST assina mensagens de tamanho m e utiliza os parâmetros k e $t = 2^\tau$ com $k\tau = m$, como no HORS. Comparado ao HORS, o HORST sacrifica o tempo de execução para reduzir o tamanho da chave pública e o tamanho combinado de uma assinatura e a chave privada. Uma chave pública de HORST é o nó raiz de uma árvore binária de hash de tamanho $\log t$, onde as folhas são elementos da chave pública de uma chave HORS. Isso reduz o tamanho da chave pública para um único valor de hash. Para isto funcionar, uma assinatura HORST não contém somente os k elementos privados mas também um caminho de autenticação por elemento de chave privada. Agora a chave pública pode ser computada dada uma assinatura. Uma assinatura completa baseada em hash inclui apenas $k \log t$ valores de hash para HORST, comparado a t valores de hash para o HORS (BERNSTEIN; HOPWOOD *et al.*, 2015).

O HORST melhora o HORS usando uma árvore binária de hash para reduzir o tamanho da chave pública de tn bits para n bits e o tamanho da assinatura e chave pública combinadas de $(k(\tau - x + 1) + 2^x)n$ bits para algum $x \in \mathbb{N}\backslash\{0\}$. O valor de x é determinado baseado em t e k tal que $k(\tau - x + 1) + 2^x$ seja mínimo. Pode acontecer que a expressão tenha seu valor mínimo para dois valores consecutivos. Neste caso, o maior valor é usado (BERNSTEIN; HOPWOOD *et al.*, 2015).

Em contraste com o esquema de assinatura único como WOTS, HORST pode ser usado para assinar mais de uma mensagem com o mesmo par de chaves. Entretanto, a cada assinatura, a segurança diminui (BERNSTEIN; HOPWOOD *et al.*, 2015). Os algoritmos para o HORST são descritos por (BERNSTEIN; HOPWOOD *et al.*, 2015):

O algoritmo de *Geração de Chave* é denotado por ($pk \leftarrow$ HORST.kg(S, Q)). Como parâmetros temos a semente $S \in \{0,1\}^n$ e a máscara de bits $Q \in \{0,1\}^{2n \times \log t}$. Primeiramente, o algoritmo computa a chave privada interna $sk = (sk_1, ..., sk_t) \leftarrow G_t(S)$. As folhas da árvore são computadas como $L_i = F(sk_i)$ para $i \in [t-1]$ e a árvore é construída usando máscaras de bits *Q*. A chave pública pk é computada como o nó raiz de uma árvore binária de altura $\log t$. A escolha das funções F e G tem relação com o algoritmo SPHINCS que será descrito na Seção 2.2.3.4, onde lá estão definidas.

O Algoritmo de Assinatura, denotado por $((\sigma, pk) \leftarrow HORST.sign(M, S, Q))$, recebe como parâmetros uma mensagem $M \in \{0,1\}^m$, semente $S \in \{0,1\}^n$ e máscara de bits $Q \in \{0,1\}^{2n \times \log t}$. Então, seja $M = (M_0, ..., M_{k-1})$ que denotam os k números obtidos dividindo M em k cadeias de tamanho $\log t$ bits cada e os interpretando como um inteiro sem sinal. A assinatura $\sigma = (\sigma_0, ..., \sigma_{k-1}, \sigma_k)$ consiste de k blocos $\sigma_i = (sk_{M_i}, Auth_{M_i})$ para $i \in [k-1]$ contento o M_i-ésimo elemento chave e o menor $\tau - x$ elementos do caminho de autenticação da folha correspondente $A_0, ..., A_{\tau-1-x}$. O bloco σ_k contém todos os 2^x nós da árvore binária no nível $\tau - x (N_{0,\tau-x}, ..., N_{2^x-1,\tau-x})$. Além da assinatura, HORST.sign também apresenta como saída a chave pública.

O Algoritmo de Verificação é denotado por $(pk' \leftarrow HORST.vf(M, \sigma, Q))$:, onde temos como parâmetros a mensagem $M \in \{0,1\}^m$, uma assinatura σ, e máscaras de bits $Q \in \{0,1\}^{2n \times \log t}$, calcula-se primeiro o M_i, como descrito acima. Então, para $i \in [k-1], y_i = \lfloor \frac{M_i}{2^{\tau-x}} \rfloor$ calcula-se $N'_{y_i,\tau-r}$ usando o Algoritmo 1 com índice $M_i, L_{M_i} = F(\sigma_i^1)$ e Auth$M_i = \sigma_i^2$. E então verifica-se que $\forall i \in [k-1] : N'_{y_i,\tau-x} = N_{y_i,\tau-x}$. Ou seja, que os nós computados batem com aqueles em σ_k. Se todas as comparações se mantiverem, usa-se σ_k para computar e então retorna-se ROOT$_0$, caso contrário retorna-se falha.

Mais informações sobre este esquema podem ser encontradas em (BERNSTEIN; HOPWOOD *et al.*, 2015).

2.2.3.3 Merkle Signature Scheme (MSS)

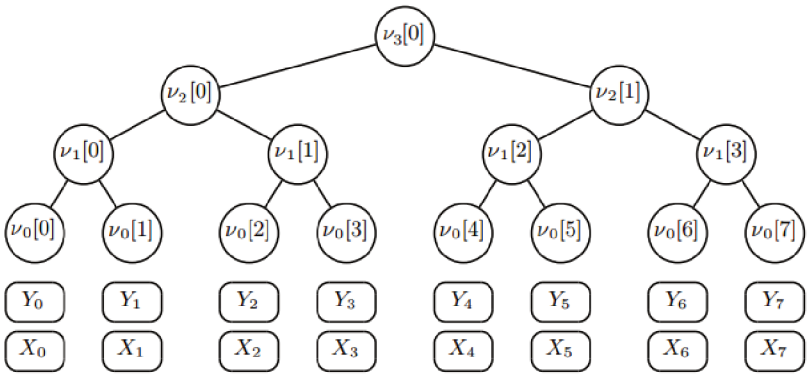
O esquema de assinatura de Merkle funciona com qualquer função de hash e esquema de assinatura única. Em (BUCHMANN; DAHMEN; SZYDLO, 2009) é descrito

como funciona o esquema. Quem for assinar uma mensagem, seleciona $H \in \mathbb{N}$, $H \geq 2$. Então, o par de chaves a ser gerado poderá assinar ou verificar 2^H documentos.

O assinante gera 2^H pares de chaves de assinatura única $(X_j, Y_j), 0 \leq j < 2^H$. X_j é a chave de assinatura e Y_j é a chave de verificação. As folhas da árvore de Merkle são os hashs $g(Y_j), 0 \leq j < 2^H$. Os nós internos da árvore de Merkle são computados de acordo com a seguinte regra de construção: um nó pai é o hash da concatenação de seus filhos à direita e à esquerda. A chave pública do MSS é a raiz da árvore de Merkle. A chave privada do MSS é a sequência de 2^H chaves de assinatura única. Denotamos os nós da árvore de Merkle por $v_h[j], 0 \leq j < 2^{H-h}$, onde $h \in \{0, ..., H\}$ é a altura do nó. Então:

$$v_h[j] = g(v_{h-1}[2j] \parallel v_{h-1}[2j+1]),\ 1 \leq h \leq H,\ 0 \leq j\ <\ 2^{H-h} \tag{24}$$

Figura 1 – Árvore de Merkle com uma altura H = 3



Fonte: Extraído de (BUCHMANN; DAHMEN; SZYDLO, 2009)

2.2.3.4 SPHINCS

SPHINCS introduz duas novas ideias que juntas reduzem drasticamente o tamanho da assinatura. Para aumentar o nível de segurança da seleção de índice aleatório, SPHINCS troca a folha do OTS (um exemplo de OTS é o LD-OTS descrito anteriormente) com um esquema FTS. No contexto do SPHINCS, isso permite poucas colisões de índices, que em troca, permite uma árvore com altura menor para o mesmo nível de segurança (BERNSTEIN; HOPWOOD *et al.*, 2015).

O SPHINCS faz uso de vários parâmetros e funções. O principal parâmetro de segurança é $n \in \mathbb{N}$. As funções incluem duas funções de hash de entrada curta $F : \{0,1\}^n \to \{0,1\}^n$ e $H : \{0,1\}^{2n} \to \{0,1\}^n$; uma função de hash com entrada arbitrária aleatória $\mathcal{H} : \{0,1\}^n \times \{0,1\}^* \to \{0,1\}^m$, para $m = poly(n)$; uma família de

geradores pseudoaleatórios $G_\lambda : \{0,1\}^n \to \{0,1\}^{\lambda n}$ para diferentes valores de λ; uma composição de famílias de funções pseudoaleatórias $\mathcal{F}_\lambda : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^n$; e uma função de famílias pseudoaleatórias $\mathcal{F} : \{0,1\}^* \times \{0,1\}^n \to \{0,1\}^{2n}$ que suporta entradas de tamanhos arbitrários. Estas funções podem ser todas construídas a partir de uma função de hash criptográfica, mas é mais natural separar as funções de acordo com suas regras (BERNSTEIN; HOPWOOD *et al.*, 2015).

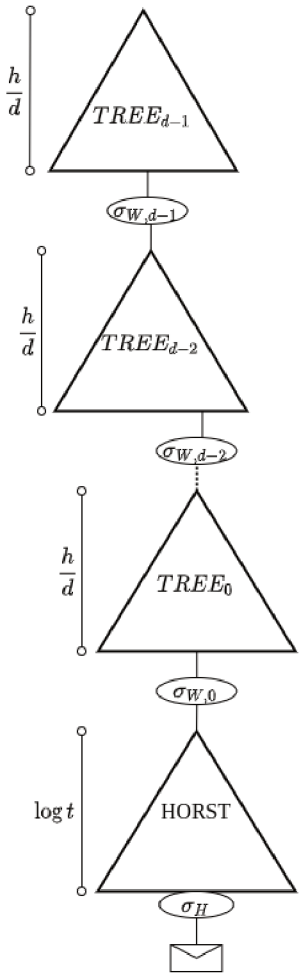
Temos, na Figura 2, a estrutura virtual de uma assinatura SPHINCS, ou seja, um caminho dentro da hiper árvore. Contém d árvores TREE$_i, i \in [d-1]$ onde cada uma possui uma árvore binária de hash que autentica os nós raízes de $2^{\frac{h}{d}}$ árvores-L na qual cada uma delas possui os nós de chave pública de um WOTS$^+$ como folhas. Cada árvore autentica a árvore abaixo usando a assinatura WOTS$^+$ $\sigma_{W,i}$. A única exceção é TREE$_0$ onde autentica uma chave pública HORST usando uma assinatura WOTS$^+$. Finalmente, o par de chaves HORST é usado para assinar a mensagem, onde o par de chaves HORST é determinado por índice gerado de forma pseudoaleatória.

Usa-se um esquema de endereçamento simples para geração de chave pseudoaleatória. Um endereço é uma cadeia de bits de tamanho $a = \lceil \log(d+1) \rceil + (d-1)\frac{h}{d} + \frac{h}{d} = \lceil \log(d+1) \rceil + h$. O endereço de um par de chaves WOTS$^+$ é obtido codificando a camada da árvore a que pertence como um registro da cadeia $\log(d+1)$-*bit*. Então, adicionando o índice da árvore na camada como uma cadeia $(d-1)(\frac{h}{d})$-*bit*. Finalmente, adicionando o índice do par de chaves WOTS$^+$ dentro da árvore codificada como uma cadeia de $\frac{h}{d}$ bits. O endereço do par de chaves HORST é obtido usando o endereço do par de chaves de WOTS$^+$ usado para assinar sua chave pública e colocando d como o valor da camada no endereço da cadeia, codificada como uma cadeia de $\lceil \log(d+1) \rceil$ bits (BERNSTEIN; HOPWOOD *et al.*, 2015).

O SPHINCS é composto por três algoritmos. O primeiro, **Algoritmo de Geração de Chaves**($(SK, PK) \leftarrow kg(1^n)$), primeiramente mostra dois valores de chaves secretas $(SK_1, SK_2) \in \{0,1\}^n \times \{0,1\}^n$. O valor de SK_1 é usado para geração de chave pseudoaleatória. O valor SK_2 é usado para gerar um índice imprevisível na assinatura e valores pseudoaleatórios para aleatorizar o hash da mensagem na assinatura. Também, p valores de n bits $Q \leftarrow \{0,1\}^{p \times n}$ uniformemente aleatórios são amostrados como máscaras de bits onde $p = max\{w-1, 2(h + \lceil \log l \rceil), 2\log t\}$. Estas máscaras de bits são usadas para todas as instâncias WOTS$^+$ e HORST como também para as árvores. Em (BERNSTEIN; HOPWOOD *et al.*, 2015), faz-se uso do Q$_{WOTS^+}$ para as primeiras $w-1$ máscaras de bit de tamanho n em Q, Q_{HORST} para o primeiro $2\log t$, Q$_{L-Tree}$ para o primeiro $2\lceil \log l \rceil$, e Q_{Tree} para as $2h$ cadeias de tamanho n em Q que segue Q_{L-Tree}.

As partes restantes do algoritmo de geração de chaves consistem na geração do nó raiz da árvore no nível $d-1$. Para este fim, os pares de chaves de WOTS$^+$ para a árvore única no nível $d-1$ são gerados. A semente para o par de chave com endereço $A = (d-1||0||i)$ onde $i \in [2^{\frac{h}{d}} - 1]$ é computado como $\mathcal{S}_\mathcal{A} \leftarrow \mathcal{F}_a(A, SK_1)$ avaliando a

Figura 2 – Estrutura virtual de uma assinatura SPHINCS



Fonte: Extraído de (BERNSTEIN; HOPWOOD *et al.*, 2015)

função de família pseudoaleatória na entrada A com chave SK_1. Então, a semente para o par de chaves WOTS$^+$ com endereço A é computado como $\mathcal{S}_\mathcal{A} \leftarrow \mathcal{F}_a(A, SK_1)$ e (BERNSTEIN; HOPWOOD *et al.*, 2015) assume que estas sementes são conhecidas por qualquer algoritmo que conhece SK_1. A chave pública do WOTS$^+$ é computada como $pk_A \leftarrow WOTS.kg(\mathcal{S}_A, Q_{WOTS^+})$. A i-ésima folha L_i da árvore é a raiz de uma árvore-L que comprime pk_A usando máscaras de bits Q_{L-Tree}. Finalmente, uma árvore binária de hash é construída usando folhas construídas e seus nós raízes se tornam PK_1. A chave secreta do SPHINCS é $SK = (SK_1, SK_2, Q)$, a chave pública é $PK = (PK_1, Q)$. O algoritmo retorna o par de chaves $((SK_1, SK_2, Q), (PK_1, Q))$.

O segundo algoritmo, **Algoritmo de Assinatura**($\sum \leftarrow sign(M, SK)$) possui como entrada uma mensagem $M \in \{0,1\}^*$ e uma chave secreta $SK = (SK_1, SK_2, Q)$, *sign* calcula uma hash da mensagem $D \in \{0,1\}^m$ aleatória. Para isto, primeiro um $R = (R_1, R_2) \in \{0,1\}^n \times \{0,1\}^n$ pseudoaleatório é calculado como $R \leftarrow \mathcal{F}(M, SK_2)$. Então,

$D \leftarrow \mathcal{H}(R_1, R_2)$ é calculado como o hash aleatório de M usando os primeiros n bits de R como aleatoriedade. Os últimos n bits de R são usados para selecionar um par de chaves HORST, calculando um bit h de índice $i \leftarrow CHOP(R_2, h)$ como os primeiros h bits de R_2.

Dado o índice i, o par de chaves HORST com endereço $A_{HORST} = (d||i(0, (d-1)\frac{h}{d})||i((d-1)\frac{h}{d}, \frac{h}{d}))$ é usado para assinar o hash da mensagem D, ou seja, os primeiros $(d-1)\frac{h}{d}$ bits de i são usados como índice de árvore e o restante dos bits são usados dentro da árvore. A assinatura HORST e a chave pública $(\sigma_H, pk_H) \leftarrow (D, S_{A_{HORST}}, Q_{HORST})$ são calculadas usando as máscaras de bits HORST e a semente $S_{A_{HORST}} \leftarrow \mathcal{F}_a(A_{HORST}, SK_1)$.

A assinatura SPHINCS $\sum = (i, R_1, \sigma_H, \sigma_{W,0}, Auth_{A_0}, \sigma_{W,d-1}, Auth_{A_{d-1}})$ contém além do índice i, aleatoriedade R_1 e a assinatura HORST σ_H também uma assinatura $WOTS^+$ e um caminho de autenticação $\sigma_{W,i}, Auth_{A_i}, i \in [d-2]$ por nível. Estes são calculados da seguinte forma: O par de chaves $WOTS^+$ com endereço A_0 é usado para assinar pk_H, onde A_0 é o endereço obtido baseado em A_{HOST} e alterando os primeiros $\lceil \log(d+1) \rceil$ bits para zero. Isto é feito executando $\sigma_{W,1} \leftarrow (pk_H, S_{A_0}, Q_{WOTS^+})$ usando as máscaras de $WOTS^+$. Então, o caminho de autenticação $Auth_{i((d-1)\frac{h}{d}, \frac{h}{d})}$ do par de chaves $WOTS^+$ é calculado. Em seguida, a chave pública $pk_{W,0}$ do $WOTS^+$ é calculada executando $pk_{W,0} \leftarrow WOTS.vf(pk_H, \sigma_{W,0}, Q_{WOTS^+})$. O nó raiz $ROOT_0$ da árvore é calculado primeiramente comprimindo $pk_{W,0}$ usando uma árvore-L. Então o algoritmo 1 é aplicado usando o índice do par de chaves $WOTS^+$ dentro da árvore, a raiz da árvore-L e $Auth_{i((d-1)\frac{h}{d}, \frac{h}{d})}$.

Este procedimento é repetido dos níveis 1 até $d-1$ com as seguintes diferenças. No nível $1 \leq j < d$, $WOTS^+$ é usado para assinar $ROOT_{j-1}$, a raiz calculada no fim da iteração anterior. O endereço do par de chaves $WOTS^+$ usado no nível j é calculado como $A_j = (j||i(0, (d-1-j)\frac{h}{d})||i((d-1-j)\frac{h}{d}, \frac{h}{d}))$, ou seja, em cada nível os últimos $\frac{h}{d}$ bits do endereço da árvore se torna um novo endereço folha e os bits restantes do endereço da árvore formada se torna um novo endereço de árvore. Por último, *sign* retorna $\sum = (i, R_1, \sigma_H, \sigma_{W,0}, Auth_{A_0}, \sigma_{W,d-1}, Auth_{A_{d-1}})$.

O terceiro algoritmo, **Algoritmo de Verificação**($b \leftarrow vf(M, \sum, SK)$) possui como entrada uma mensagem $M \in \{0,1\}^*$, uma assinatura $\sum$ e uma chave pública PK, o algoritmo calcula o hash da mensagem $D \leftarrow \mathcal{H}(R_1, M)$ usando aleatoriedade R_1 contido na assinatura. O hash da mensagem D e as máscaras Q_{HORST} de $HORST$ da PK são usadas para calcular a chave pública *HORST* $pk_H \leftarrow HORST.vf(D, \sigma_H, Q_{HORST})$ da assinatura *HORST*. Se *HORST.vf* retornar falha, a verificação retorna como falso. A chave pública *HORST* por sua vez é usada junto com as máscaras de bit $WOTS^+$ e a assinatura $WOTS^+$ para calcular a primeira chave pública $pk_{W,0} \leftarrow WOTS.vf(pk_H, \sigma_{W,0}, Q_{WOTS^+})$ de $WOTS^+$. Uma árvore-L é usada para calcular $L_{i((d-1)\frac{h}{d}, \frac{h}{d})}$, a folha correspondente a $pk_{W,0}$. Então, a raíz $ROOT_0$ da respectiva árvore é calculada usando o

Algoritmo 1 com índice $i((d-1)\frac{h}{d}, \frac{h}{d}$, folha $L_{i((d-1)\frac{h}{d},\frac{h}{d}}$ e caminho de autenticação $Auth_0$.

Algorithm 1: Computação da Raiz (extraído de (BERNSTEIN; HOPWOOD *et al.*, 2015))

Input: índice de folha i, folha L_i, caminho de autenticação $\text{Auth}_i = (A_0, ..., A_{h-1})$ para L_i.

Output: nó raiz ROOT da árvore que contém L_i.

$P_0 \leftarrow L_i$;

for $j \leftarrow 1$ *até h* **do**

$$P_j = \begin{cases} H((P_{j-1}||A_{j-1}) \bigoplus Q_j), se \lfloor \frac{i}{2^{j-1}} \rfloor \equiv 0 \mod 2 \\ H((A_{j-1}||P_{j-1}) \bigoplus Q_j), se \lfloor \frac{i}{2^{j-1}} \rfloor \equiv 1 \mod 2 \end{cases}$$

end

Result: P_h

Então, este procedimento é repetido dos níveis 1 até $d-1$ com as seguintes diferenças. Primeiramente, no nível $1 \leq j < d$, a raiz da árvore previamente processada $ROOT_{j-1}$ é usada para calcular a chave pública $pk_{W,j}$ de $WOTS^+$. Então, a folha calculada de $pk_{W,j}$ usando uma árvore-L é $L_{i((d-1-j)\frac{h}{d},\frac{h}{d}}$. Ou seja, o índice da folha dentro da árvore pode ser calculada cortando os últimos $j(\frac{h}{d})$ bits de i e então usando os últimos $\frac{h}{d}$ bits da cadeia de bits resultante.

O resultado da repetição final no nível $d-1$ é um valor $ROOT_{d-1}$ para o nó raiz de uma única árvore no nível mais alto. Este valor é comparado com o primeiro elemento da chave pública, ou seja, $PK_1 \stackrel{?}{=} ROOT_{d-1}$. Se a comparação for mantida, *vf* retorna verdadeiro, se não retorna falso.

Em (BERNSTEIN; HOPWOOD *et al.*, 2015) é proposta uma instância específica do SPHINCS chamada de SPHINCS-256. Esta instância tem os seguintes parâmetros

Parâmetro	**Valor**
n	256
m	512
h	60
d	12
w	16
t	2^{16}
k	32

Esta instância SPHINCS-256, apresenta uma chave pública de 1,056KB, uma chave privada de 1,088KB e uma assinatura de 41KB. O SPHINCS não utiliza rastreamento de estado, portando é sem estado e construído em um esquema de poucas assinaturas. Uma árvore de Merkle é utilizada e, ao assinar uma mensagem, uma chave índice de poucas assinaturas é selecionada aleatoriamente. Desta forma, as chaves pública e privada resultantes ficam em torno de 1KB e a assinatura da mensagem fica em torno de 41KB - na qual é significantemente menor do que as assinaturas da

maioria dos esquemas baseados em hash. O desempenho também é melhor para geração de assinatura e verificação (BUTIN, 2017).

2.2.4 Criptografia baseada em reticulados

Em (NEJATOLLAHI *et al.*, 2019) define-se um reticulado $\mathcal{L}$ como um conjunto discreto infinito de pontos no espaço Euclidiano de dimensão n com uma estrutura periódica. Uma base (B) de um reticulado é definido como $b_1, b_2, ..., b_n \in \mathbb{R}^{d \times n}$ e possui n vetores linearmente independentes em $\mathbb{R}^d$. B é uma matriz $d \times n$ na qual a iésima coluna é o vetor b_i tal que $B = [b_1, b_2, ..., b_n]$. Inteiros n e d são a classificação e dimensão do reticulado $\mathcal{L}(B)$. Se $n = d$, $\mathcal{L}(B)$ é um reticulado de classificação (ou dimensão) completa em $\mathbb{R}^d$. Todas as combinações inteiras geradas pela base da matriz B formam o reticulado $\mathcal{L}$:

$$\mathcal{L}(B) = Bx : x \in \mathbb{Z}^n = \mathcal{L}(b_1, b_2, ..., b_n) = \left\{ 1 \sum_{i=1}^{n} x_i b_i : x_i \in \mathbb{Z}, 1 \leq i \leq n \right\} \quad (25)$$

Segundo (PEIKERT, 2016), criptografia baseada em reticulado é o uso de problemas conjecturados difíceis em reticulados em $\mathbb{R}^n$ como a fundação para sistemas criptográficos seguros. Características atrativas de criptografia de reticulados incluem aparente resistência a ataques quânticos, alta eficiência assintótica e paralelismo, e soluções de longa data em criptografia.

Em (PEIKERT, 2016) alguns recursos da criptografia baseada em reticulados são descritos. O primeiro a ser descrito é a segurança conjecturada contra ataques quânticos onde não há algoritmos quânticos eficientes para resolver os problemas tipicamente usados em criptografia baseada em reticulados.

Outro recurso apontado por (PEIKERT, 2016) é a simplicidade do algoritmo, eficiência e o paralelismo. Sistemas de criptografia baseados em reticulados possuem um algoritmo frequentemente simples e altamente paralelizável, consistindo principalmente de operações lineares em vetores e matrizes com módulo inteiro relativamente pequeno. Além disso, construções baseados em reticulados algébricos sobre certos anéis podem ser especialmente eficientes, e em alguns casos superar os sistemas mais tradicionais por uma margem significativa.

O terceiro recurso descrito por (PEIKERT, 2016) é a forte garantia de segurança. Criptografia requer intratabilidade no caso médio, isto é, problemas no qual instâncias aleatórias são difíceis de resolver.

O último recurso apontado por (PEIKERT, 2016), é sobre construções poderosas e versáteis de objetos criptográficos. Construções criptográficas baseadas em reticulados mantém uma grande promessa para a criptografia pós-quântica pois eles desfrutam de provas de segurança muito fortes, implementações relativamente eficientes, grande simplicidade e até competem com as melhores alternativas conhecidas.

Muitas destas construções criptográficas são bastante eficientes, e algumas até competem com as melhores alternativas conhecidas. Eles são tipicamente bastante simples para implementar e acredita-se que todos sejam seguros contra computadores quânticos (MICCIANCIO; REGEV, 2009).

Construções criptográficas baseadas em reticulados são baseadas na dificuldade presumida de problemas de reticulados, dos quais o mais básico é o Shortest Vector Problem[2] (SVP). Aqui, damos como parâmetro um reticulado representado por uma base arbitrária e nosso objetivo é obter o menor vetor diferente de zero. De fato, normalmente se considera a variante de aproximação do SVP onde o objetivo é gerar um vetor de reticulado cujo comprimento é no máximo algum fator de aproximação $\gamma(n)$ vezes o comprimento do menor vetor diferente de zero, onde n é a dimensão do reticulado (MICCIANCIO; REGEV, 2009).

O mais conhecido e amplamente estudado algoritmo para problemas de reticulados é o algoritmo LLL, desenvolvido em 1982 por Lenstra, Lenstra e Lovász. Este é um algoritmo de tempo polinomial para SVP que alcança um fator de aproximação de $2^{O(n)}$ onde n é a dimensão do reticulado. Por pior que pareça o algoritmo LLL é surpreendentemente útil, com aplicações que variam desde a fatoração de polinômios sobre números racionais, programação inteira, bem como muitas aplicações em análise criptográficas (MICCIANCIO; REGEV, 2009).

Uma garantia de segurança forte é uma das características distintivas da criptografia baseada em reticulados. Praticamente, todas as outras construções criptográficas são baseadas na dificuldade do caso médio. Por exemplo, quebrar um sistema de criptografia baseado na fatoração pode implicar a habilidade de fatorar alguns números escolhidos de acordo com uma certa distribuição, mas não na habilidade de fatorar todos os números (MICCIANCIO; REGEV, 2009).

Problemas em reticulados são tipicamente bem difíceis. Os melhores algoritmos conhecidos rodam em tempo exponencial, ou fornecem taxas de aproximação muito ruins. O campo da criptografia baseada em reticulados se desenvolveu baseado na suposição que reticulados são difíceis (MICCIANCIO; REGEV, 2009). Ainda em (MICCIANCIO; REGEV, 2009) afirma-se que é provável que criptografia baseada em reticulados seja adequada para um mundo pós-quântico e que também atualmente não há algoritmos quânticos conhecidos para resolver problemas em reticulados que desempenham significantemente melhor do que o melhor algoritmo clássico.

As tentativas de resolver problemas de reticulados por algoritmos quânticos foram feitas desde o algoritmo de Shor (MICCIANCIO; REGEV, 2009). A principal dificuldade é que a técnica para encontrar a periodicidade, que é usada no algoritmo de fatoração de Shor e relacionada a algoritmos quânticos, não parece ser aplicável para problemas de reticulados. Com isso, (MICCIANCIO; REGEV, 2009) descreve outra conjectura na

[2] Problema do menor vetor

qual diz que não há algoritmo quântico de tempo polinomial que aproxima problemas de reticulados dentro dos fatores polinomiais. Com isso, o mesmo deixa claro que esta conjectura não pode ser interpretada da forma que algoritmos quânticos não tenham influência no entendimento de problemas de reticulados.

2.2.4.1 Sistema de Criptografia de Ajtai-Dwork

Ajtai deu a primeira redução de pior caso para caso médio em problemas de reticulados e também o primeiro objeto criptográfico com uma prova de segurança assumindo a dificuldade dos bem estudados problemas em reticulados. O trabalho de Ajtai deu a primeira função criptográfica baseada na premissa de complexidade de caso médio de qualquer tipo. Também, introduziu o problema Short Integer Solution (SIS) e a sua função de via única associada, e provou que resolver o problema é tão difícil quanto aproximar vários problemas de reticulados de pior caso (PEIKERT, 2016).

Em (AJTAI; DWORK, 1997) apresenta-se o sistema de criptografia de chave pública probabilístico no qual é seguro, a menos que o pior caso do problema de reticulados, descrito a seguir, possa ser resolvido em tempo polinomial: "Encontre o menor vetor não nulo em um reticulado L de dimensão n onde o menor vetor v é único no sentido que qualquer outro vetor no qual o tamanho é ao menos $n^c\|v\|$ é paralelo a v".

Como um primeiro esquema de criptografia com uma prova de segurança sob uma suposição de complexidade de pior caso, Ajtai-Dwork foi um avanço teórico. Entretanto, do ponto de vista prático, tem alguns recuos significantes: suas chaves públicas são do tamanho $\tilde{\mathrm{O}}(n^4)$, e as chaves privadas são de tamanho $\tilde{\mathrm{O}}(n^2)$, tornando-o inviável para uso (PEIKERT, 2016).

Segundo (AJTAI; DWORK, 1997), uma instância desse sistema de criptografia é uma coleção de hiperplanos ocultos, no qual formam a chave privada, junto com um método para gerar um ponto perto a um dos hiperplanos na coleção, formando a chave pública. A chave pública é escolhida de forma a não revelar a coleção de hiperplanos e, dada a chave pública, qualquer habilidade de descobrir a coleção implica na habilidade de resolver o problema de pior caso do menor vetor único.

Seja um grande cubo $\mathcal{Q} \in \mathbb{R}^n$ com n dimensões. A encriptação é realizada bit a bit: zero é encriptado usando a chave pública para encontrar um vetor aleatório $v \in \mathcal{Q}$ perto de um dos hiperplanos, onde o texto cifrado é v; um é encriptado escolhendo um vetor u aleatório uniformemente de $\mathcal{Q}$, onde o texto cifrado é simplesmente u. O deciframento de um texto cifrado x é realizado usando a chave privada para determinar a distância de x para o hiperplano oculto mais próximo. Se a distância for suficientemente pequena, então x é decifrado como zero; caso contrário x é decifrado como um. Há uma pequena probabilidade polinomial de ocorrer um erro no deciframento: um ciframento de um pode ser decifrado como zero.

2.2.4.2 NTRU

O esquema de criptografia NTRU é considerado eficiente, possui chaves muito compactas, e tem resistido a esforços critoanalíticos significativos quando apropriadamente parametrizados. Ao contrário do Ajtai-Dwork, há relativamente pouco entendimento do esquema NTRU e seus problemas computacionais de caso médio associados. Em particular, não há redução conhecida de qualquer problema de pior caso de reticulados para qualquer versão do problema NTRU, nem para quebrar a segurança semântica do sistema de criptografia (PEIKERT, 2016).

O procedimento de criptografia usa um sistema de mistura baseado em álgebra polinomial e redução módulo dois números p e q, enquanto que o procedimento para decifrar um sistema sem mistura onde a validade depende da teoria da probabilidade elementar. A segurança do esquema de criptografia de chave pública NTRU vem da interação de um sistema de mistura polinomial com a interdependência de redução módulo p e q. A segurança também conta com o fato que para a maioria dos reticulados, é muito difícil encontrar vetores extremamente curtos (HOFFSTEIN; PIPHER; SILVERMAN, 1998).

O NTRU se encaixa em um framework geral de um sistema de criptografia probabilístico. Isso significa que o processo de criptografia inclui um elemento aleatório, então cada mensagem tem muitos textos cifrados possíveis. Para cifrar e decifrar é extremamente rápido e possui criação de chave rápido e fácil (HOFFSTEIN; PIPHER; SILVERMAN, 1998).

Uma notação do algoritmo é mostrada em (HOFFSTEIN; PIPHER; SILVERMAN, 1998). Um sistema de criptografia NTRU depende de três parâmetros inteiros (N, p, q) e quatro conjuntos L_f, L_g, L_ϕ, L_m de polinômios de grau $N-1$ com coeficientes inteiros. Uma observação feita pelos autores é que p e q não precisam ser primos, mas assumem que o são, e q sempre será consideravelmente maior que p. Tudo isso funciona no anel $R = \frac{\mathbb{Z}[X]}{X^N-1}$. Um elemento $F \in R$ será escrito como um polinômio ou vetor:

$$F = \sum_{i=0}^{N-1} F_i x_i = [F_0, F_1, ..., F_{N-1}] \tag{26}$$

Se escreve $\circledast$ para denotar a multiplicação em R. A multiplicação estrela é dada explicitamente como um produto da convolução cíclica:

$$F \circledast G = H \; com \; H_k = \sum_{i=0}^{k} F_i G_{k-i} + \sum_{i=k+1}^{N-1} F_i G_{N+k-i} = \sum_{i+j \equiv k (mod N)} F_i G_j \tag{27}$$

A computação de um produto $F \circledast G$ requer N^2 multiplicações. Entretanto, para um produto típico utilizado pelo NTRU, G ou F possui coeficientes pequenos, então a computação de $F\circledast$ é muito rápida (HOFFSTEIN; PIPHER; SILVERMAN, 1998).

(HOFFSTEIN; PIPHER; SILVERMAN, 1998) mostra o processo de criação de chaves. Para criar uma chave, Bob seleciona aleatoriamente 2 polinômios $f, g \in L_g$. O polinômio f deve possuir inversa módulo q e módulo p. Para a escolha de parâmetros adequados, isso será verdade para a maioria das escolhas de f, e a computação atual dessas inversas é realizada utilizando uma modificação do algoritmo de Euclides. As inversas são descritas como F_q e F_p, ou seja:

$$F_q \circledast f \equiv 1 \mod q \; e \; F_p \circledast f \equiv 1 \mod q \tag{28}$$

Em seguida, Bob calcula a quantidade:

$$h \equiv F_q \circledast g \mod q \tag{29}$$

A chave pública de Bob é o polinômio h. A chave privada de Bob é o polinômio f, apesar que na prática ele armazenará F_p.

Os processos de cifragem e decifragem também são descritos em (HOFFSTEIN; PIPHER; SILVERMAN, 1998). No processo de cifragem, vamos supor que Alice queira mandar uma mensagem a Bob. Ela começa selecionando uma mensagem m do conjunto de textos planos L_m. Em seguida, ela aleatoriamente escolhe um polinômio $\phi \in L_\phi$ e utiliza a chave pública de Bob h para calcular:

$$e \equiv p\phi \circledast h + m \mod q \tag{30}$$

Esta é a mensagem encriptada onde Alice transmite a Bob. No processo de decifragem, vamos supor que Bob recebe a mensagem e de Alice e queira decifrá-la usando sua chave privada f. Para fazer isto eficientemente, Bob deve calcular o polinômio F_p. Para decifrar e, Bob primeiro calcula:

$$a \equiv f \circledast e \mod q \tag{31}$$

onde ele escolhe os coeficientes de a no intervalo de $\frac{-q}{2}$ a $\frac{q}{2}$. Agora, tratando a como um polinômio com coeficientes inteiros, Bob recupera a mensagem calculando:

$$F_p \circledast a \mod p \tag{32}$$

Implementações práticas do esquema NTRU são descritas como caso de segurança moderada, alta segurança e mais alta segurança em (HOFFSTEIN; PIPHER; SILVERMAN, 1998).

2.2.5 Criptografia polinomial multivariada

Estes esquemas são baseados na dificuldade de resolver sistemas polinomiais multivariados sobre corpos finitos. Vários sistemas criptográficos multivariáveis foram propostos nas últimas décadas, e muitos foram quebrados. A criptografia multivariada tem sido historicamente mais bem sucedida como uma abordagem para assinaturas (CHEN, L. *et al.*, 2016). Esta família é considerada como uma das principais famílias de criptografia de chave pública que poderia potencialmente resistir aos computadores quânticos do futuro (DING; YANG, 2009).

Sistemas de assinatura multivariados baseados em polinômios oferecem o benefício de assinaturas curtas e geralmente eficientes. Baseados em operações aritméticas simples, eles são rápidos em relação a outros tipos de esquemas de assinatura. Entretanto, as chaves públicas tendem a ser comparativamente grandes. Isso pode ser problemático para dispositivos embarcados (BUTIN, 2017).

Sua segurança é baseada na dificuldade de resolver conjuntos de equações não lineares sobre um corpo finito e está relacionado a estrutura de polinômios ideais (BUTIN, 2017).

A segurança de esquemas multivariados é baseada no fato que resolver um conjunto de equações polinomiais (geralmente) quadráticas multivariáveis e é um problema que foi provado ser NP-Completo sobre qualquer campo e acredita-se que na média seja difícil para computadores clássicos e quânticos. Até agora, não há evidências que revelam que computadores quânticos poderiam resolver este tipo de problema eficientemente. Este problema também é chamado de MQ (Multivariate Quadratic[3]) (DING; YANG, 2009; XIN WANG, 2019). Esquemas de criptografia de chave pública multivariável são em geral muito mais eficientes do que o RSA (Seção 2.1) em computação (CHEN, J. *et al.*, 2019).

Por razões de eficiência, a chave pública dos sistemas de criptografia de chaves pública polinomial multivariável é geralmente um sistema de polinômios quadráticos em várias variáveis sobre um corpo finito K com q elementos (DING; YANG, 2009).

Um exemplo é o esquema de assinatura Rainbow, para o qual parâmetros típicos para o campo subjacente F_{31}, produzem uma chave pública de 45 Kbytes e uma chave privada de 31 Kbytes. Uma troca entre o tempo de geração de um par de chaves e tamanho está disponível para o Rainbow: chaves e assinaturas podem ser menores se aceitarmos uma geração de chave mais lenta. A prova da segurança comprovada de esquemas de assinatura baseados em polinômios multivariáveis não é claro no momento, resultando em incerteza indesejável (BUTIN, 2017).

Como previsto por Diffie e Hellmann, um sistema de chave pública depende da existência de uma classe de "função alçapão". Esta classe e a estrutura matemática por trás irá determinar todas as características essenciais da PKC. Criptografia de

[3] Quadrático Multivariável

chave pública multivariável é o estudo de PKCs onde a função alçapão toma a forma de um mapa polinomial quadrático multivariado sobre um corpo finito (DING; YANG, 2009).

(CHEN, J. *et al.*, 2019) descreve que um esquema de criptografia de chave pública multivariável no corpo finito F_q é usualmente construído como:

$$P = L_1 \circ F \circ L_2 \tag{33}$$

onde F é um conjunto de m equações quadráticas polinomiais multivariáveis com n variáveis e L_1 é uma transformação de $\mathbb{F}_q^m$ para $\mathbb{F}_q^m$ e L_2 é uma transformação de $\mathbb{F}_q^n$ para $\mathbb{F}_q^n$. Um esquema de assinatura MPKC possui os algoritmos: **GeracaoDeChaves**, **Assinar**, **Verificar**.

Para um esquema de assinatura MPKC o algoritmo de configuração Configuracao(1^λ), recebe 1^λ como parâmetro, e retorna o parâmetro de sistema p, no qual contém (n, m, q) e todas as operações aritméticas a seguir são sobre esse corpo finito (CHEN, J. *et al.*, 2019).

O algoritmo de geração de chaves GeracaoDeChaves(p) recebe o parâmetro *p* e então retorna $pk = \overline{F}$ e $sk = (L_1, F, L_2)$ (CHEN, J. *et al.*, 2019).

O algoritmo de assinatura Assinar(M, L_1, F, L_2) e assumindo que o documento que precisa ser assinado é $M = (y_1, y_2, ..., y_m)$. Primeiro, o usuário que tem acesso à chave privada calcula $M = L_1^{-1}(M)$, e então resolve a equação $M = F(X)$ e descobre X. Finalmente, computa $S = L_2^{-1}(X)$ (CHEN, J. *et al.*, 2019).

O algoritmo de verificação Verifica(σ, M, P) retorna *um* se $\overline{F}(\sigma) = M$, se não retorna *zero*. Para maiores informações sobre esse esquema consulte (CHEN, J. *et al.*, 2019).

2.2.5.1 Rainbow

Rainbow é um esquema de assinatura assimétrico e pertence a uma família de esquemas assimétricos multivariáveis, uma promissora sub área da criptografia de chave pública onde o desenvolvimento foi motivado pela busca de alternativas ao RSA. Rainbow é sistema de Oil-Vinegar multinível. A importância do trabalho feito por (DING; SCHMIDT, 2005) está na potencial aplicação do sistema Rainbow como um sistema de autenticação de chaves públicas muito eficiente. Em vez de utilizar a dificuldade de resolver uma equação de variável única sob um grande anel finito como o caso do RSA e logaritmos discretos como no DSA, esquemas multivariáveis exploram a dificuldade de resolver um sistema de equações multivariáveis sob um pequeno corpo finito (BILLET; GILBERT, 2006).

O esquema de assinatura Rainbow foi quebrado recentemente e não recomendamos o seu uso `https://eprint.iacr.org/2022/214.pdf`. A seguir descrevemos o seu funcionamento.

Em (DING; SCHMIDT, 2005) faz-se as seguintes definições. Seja S o conjunto $\{1, 2, 3, ..., n\}$. Seja v_1, v_2, v_u u elementos inteiros tal que $0 < v_1 < v_2 ... < v_u = n$, e definido o conjunto de inteiros $S_l = \{1, 2, ..., v_l\}$ para $l = 1, ..., u$ de forma que temos:

$$S_1 \subset S_2 \subset ... \subset S_u = S$$

O número de elementos em S_i é v_i. Seja

$$o_i = v_{i+1} - v_i, \ para \ i = 1, ..., u - 1$$

Seja O_i o conjunto tal que

$$O_i = S_{i+1} - S_i, \ para \ i = 1, ..., u - 1$$

Seja P_l o espaço linear de polinômios quadráticos da forma

$$\sum_{i \in O_l, \ j \in S_l} \alpha_{i,j} x_i x_j + \sum_{i,j \in S_l} \beta_{i,j} x_i x_j + \sum_{i, \in S_{l+1}} \gamma_{i,j} x_i + \eta$$

São tipos de polinômios de *Oil* e *Vinegar* tal que $x_i, i \in O_l$ são variáveis *Oil* e $x_i, i \in S_l$ são variáveis *Vinegar*. Chama-se $x_i, i \in O_l$ uma variável *Oil* de l-ésima camada e $x_i, i \in S_l$ uma variável *Vinegar* de l-ésima camada. Chama-se qualquer polinômio em P_l uma l-ésima camada um polinômio *Oil* e *Vinegar* (DING; SCHMIDT, 2005).

Em (DING; SCHMIDT, 2005) explica-se que cada $P_l, l = 1, ..., u - 1$ é um conjunto de polinômios *Oil* e *Vinegar*. Cada polinômio em P_l possui $x_i, i \in O_i$ como suas variáveis *Oil* e $x_i, i \in S_i$ como suas variáveis *Vinegar*. Os polinômios *Oil* e *Vinegar* em P_i podem ser definidos como polinômios tal que $x_i, i \in O_i$ são as variáveis *Oil* e $x_i, i \in S_i$ são as variáveis *Vinegar*. Isso pode ser ilustrado pelo fato que

$$S_{i+1} = S_i, O_i$$

A transformação F do esquema é definido por (DING; SCHMIDT, 2005). É um mapa F de K^n até k^{n-v_1} tal que

$$\begin{aligned} F(x_1, ..., x_n) &= (\widetilde{F}_1(x_1, ..., x_n), ..., \widetilde{F}_{u-1}(x_1, ..., x_n)) \\ &= (F_1(x_1, ..., x_n), ..., F_{n-v_1}(x_1, ..., x_n)) \end{aligned}$$

onde cada $\widetilde{F}_i$ consiste de o_i polinômios quadráticos escolhidos randomicamente a partir de P_i. Por polinômios randomicamente escolhidos se quer dizer que seus coeficientes são escolhidos aleatoriamente.

Em (DING; SCHMIDT, 2005) afirma-se que desta forma podemos ver que *F* na realidade tem $u-1$ camadas de construções de *Oil* e *Vineger*. O primeiro nível consiste de o_1 polinômios $F_1, ..., F_{o_1}$ tal que $x_j, j \in O_1$ são as variáveis *Oil* e $x_j, j \in S_1$ são as

variáveis *Vineger*. A i-ésima camada consiste de o_i polinômios $F_{v_i+1}, ..., F_{v_i+1}$ tal que $x_j, j \in O_i$ são as variáveis *Oil* e $x_j, j \in S_i$ são as variáveis *Vineger*. A partir disso, podemos construir

$$
\begin{aligned}
[x_1, ..., x_{v_1}]; \{x_{v_1+1}, ..., x_{v_2}\} \\
[x_1, ..., x_{v_1}, x_{v_1+1}, ..., x_{v_2}]; \{x_{v_2+1}, ..., x_{v_3}\} \\
[x_1, ..., x_{v_1}, x_{v_1+1}, ..., x_{v_2}, x_{v_2+1}, ..., x_{v_3}]; \{x_{v_3+1}, ..., x_{v_4}\} \\
...; ... \\
[x_1, ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., x_{v_u-1}]; \{x_{v_u-1}, ..., x_{v_n}\}
\end{aligned}
$$

Cada linha acima representa uma camada do Rainbow. Para cada l-ésima camada acima, os valores entre [ ] são as variáveis *Vinegar* e aquelas entre { } são as variáveis *Oil* e a variável *Vinegar* de cada camada consiste de todas as variáveis do nível anterior (DING; SCHMIDT, 2005).

(DING; SCHMIDT, 2005) chamam de F um mapa polinomial Rainbow com $u - 1$ camadas. Ainda definem L_1 e L_2 duas transformações lineares afins inversíveis escolhidos aleatoriamente. Seja

$$\overline{F}(x_1, ..., x_n) = L_1 \circ F \circ L_2(x_1, ..., x_n)$$

no qual consiste de $n - v_1$ polinômios quadráticos com n variáveis.

Para um esquema de assinatura Rainbow, a chave pública consiste de $n - v_1$ componentes polinomiais de $\overline{F}$ e a estrutura do corpo de k. A chave privada consiste das transformações L_1, L_2 e F (DING; SCHMIDT, 2005).

Ainda em (DING; SCHMIDT, 2005) é descrito o processo de assinatura. Para assinar um documento, que é um elemento $Y' = (y'_1, .., y'_{n-v_1})$ em k^{n-v_1}, é preciso encontrar a solução da equação

$$L_1 \circ F \circ L_2(x_1, ..., x_n) = \overline{F}(x_1, ..., x_n) = Y'$$

Podemos aplicar a inversa de L_1, então temos

$$F \circ L_2(x_1, ..., x_n) = L_1^{-1} Y' = \overline{Y}'$$

Em seguida, precisamos inverter F. Neste caso, precisamos resolver a equação

$$F(x_1, ..., x_n) = \overline{Y}' = (\overline{y}'_1, .., \overline{y}'_{n-v_1})$$

Primeiro randomicamente escolhemos os valores de $x_1, ..., x_{v_1}$ e os colocamos na primeira camada das o_1 equações dadas por

$$\widetilde{F}_1 = (\overline{y}'_1, .., \overline{y}'_{o_1})$$

Isso produz um conjunto de o_1 equações lineares com o_1 variáveis $x_{o_1+1}, ..., x_{v_2}$, que resolvemos para encontrar os valores de $x_{o_1+1}, ..., x_{v_2}$. Então temos todos os valores de $x_i \in S_2$.

Então, colocamos estes valores na segunda camada de polinômios, que novamente irá produzir o_2 números de equações lineares, que então nos dará todos os $x_i, i \in S_3$. Repetimos este procedimento até encontramos a solução (DING; SCHMIDT, 2005).

Se a qualquer momento, um conjunto de equações lineares não tiver solução, iremos começar novamente do início escolhendo outro conjunto de valores para $x_1, ..., x_{v_1}$. Este passo se repetirá até que uma solução seja encontrada (DING; SCHMIDT, 2005).

Novamente (DING; SCHMIDT, 2005), nos mostra como verificar uma assinatura. Só precisamos checar se de fato

$$\widetilde{F}(X') = Y'.$$

Um exemplo de uso é mostrado em (DING; SCHMIDT, 2005), onde os autores fazem a escolha de k para ser um corpo finito de tamanho $q = 2^8$.

Seja $n = 33$ e S o conjunto $\{1, 2, 3, ..., 33\}$.

Seja $u = 5$ e $v_1 = 6, v_2 = 12, v_3 = 17, v_4 = 22, v_5 = 33$.

Também temos $o_1 = 6, o_2 = 5, o_3 = 5, o_4 = 11$.

Neste caso, ambos $\overline{F}$ e F são mapas de k^{33} para k^{27}.

A chave pública consiste de 27 polinômios quadráticos com 33 variáveis. O número total de coeficientes para a chave pública é $27 \times 34 \times \frac{35}{2} = 16,065$, ou cerca de 15KB de armazenamento.

A chave privada consiste de 11 polinômios com 22 variáveis *Vinegar* e 11 variáveis *Oil*, 5 polinômios com 17 variáveis *Vinegar* e 5 variáveis *Oil*, 5 polinômios com 12 variáveis *Vinegar* e 5 variáveis *Oil*, e 6 polinômios com 6 variáveis *Vinegar* e 6 variáveis *Oil* mais as duas transformações lineares afins L_1 e L_2. O tamanho total é cerca de 10KB. Este esquema de assinatura assina um documento de tamanho $8 \times 27 = 216$ bits com uma assinatura de $8 \times 33 = 264$ bits.

Mais informações sobre o esquema Oil-Vinegar podem ser encontradas em (DING; PETZOLDT, 2017).

3 COMPARATIVO ENTRE AS TÉCNICAS

Este capítulo apresenta as principais características dos algoritmos pós-quânticos presentes na terceira rodada do NIST bem como mostra um conjunto de testes de desempenho dos mesmos e suas variações. As implementações dos algoritmos foram baixadas do site do NIST no endereço `https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions`. Fizemos os mesmos testes de desempenho com o algoritmo RSA e comparamos com os algoritmos pós-quânticos propostos. A ideia foi fazer uma comparação dos algoritmos colocando-os lado a lado com informações sobre tamanho de chaves e assinatura, tipo e categoria de forma que fique mais fácil visualizar estas características.

Em (BALAMURUGAN *et al.*, 2021) é apresentado um comparativo com os algoritmos pós-quânticos predominantes onde são descritas informações como tamanho de chaves, assinatura, pontos fortes e fracos, conforme apresentado abaixo. No quadro a seguir descreve-se a categoria de cada algoritmo.

Quadro 6 – Algoritmos pós-quânticos, suas categorias e tipos

Algoritmo	**Categoria**	**Tipo de Algoritmo**
Mc Eliece Classic	Criptografia Baseada em Código	Criptografia
SPHINCS	Criptografia Baseada em Hash	Assinatura
SPHINCS+	Criptografia Baseada em Hash	Assinatura
NTRU Encrypt	Criptografia Baseada em Reticulado	Criptografia
Rainbow	Criptografia Polinomial Multivariada	Assinatura

Ainda, apresentamos o tamanho de chaves e assinatura baseado em (BALAMURUGAN *et al.*, 2021).

Quadro 7 – Algoritmos pós-quânticos e tamanhos de chaves e assinaturas

Algoritmo	**Chave Pública**	**Chave Privada**	**Assinatura**
Mc Eliece Classic	1MB	11,5KB	-
SPHINCS	1KB	1KB	41KB
SPHINCS+	32B	64B	8KB
NTRU Encrypt	6,13KB	6,743KB	-
Rainbow	124KB	95KB	41KB

O algoritmo Mc Eliece Classic, é um dos algoritmos de criptografia que foi bem sucedido até a terceira rodada do NIST. Porém seu ponto fraco se deve ao fato de apresentar tamanhos de chaves muito grandes.

No caso do SPHINCS, seus pontos fortes estão entre ser a melhor alternativa para assinatura, apresentar assinatura e tamanho de chaves pequenos. Seu ponto fraco é ter apenas um esquema de assinatura disponível e não um sistema de criptografia completo. No caso do SPHINCS+, seu ponto fraco é a velocidade.

O NTRU Encrypt apresenta uma melhor eficiência para cifrar e decifrar nas implementações tanto em hardware quanto em software. Possui uma geração de chave muito mais rápida e apresenta baixo consumo de memória. Entre seus pontos fracos estão a alta complexidade e a possibilidade de ocorrer uma falha de decifração.

O algoritmo Rainbow foi removido dos testes porque o mesmo foi quebrado recentemente (BEULLENS, 2022).

3.1 TESTES

Os testes foram realizados em um computador com Ubuntu 20.04.3 LTS, processador AMD® Ryzen 7 5800x 8 núcleos e 16 threads, 32GB de memória e uma placa de vídeo NVDIA GTX 1650 utilizando o driver NV166. Todos os dados dos gráficos utilizados a seguir estão presentes no Apêndice A.

Foram realizados testes para o algoritmo RSA e os algoritmos pós-quânticos McEliece, SPHINCS e NTRU da terceira rodada encontrados em (NIST, 2017d). Cada teste, tanto de geração de chave como de assinatura, foi executado 10 vezes a fim de gerar uma média.

Para todos os algoritmos foram realizados testes de geração de chaves a fim de obter o tempo médio para gerar uma chave. Neste processo, pode-se descobrir o tamanho das chaves públicas e privadas. Assim, a partir do tempo médio de geração de chaves, podemos calcular a quantidade média de chaves geradas a cada segundo. É importante destacar que quando falamos sobre chaves, nos referimos a um par de chaves (chave pública e chave privada).

A partir da quantidade de chaves geradas a cada segundo, juntamente com o tamanho das chaves, podemos trazer a ideia de armazenamento médio. Quando falamos sobre espaço de armazenamento, consideramos que estas chaves serão armazenadas em algum sistema de armazenamento. Com isso, multiplicando a quantidade média de chaves geradas e o tamanho delas (lembrando de multiplicar o tamanho da chave pública e privada separadamente e somá-los depois), podemos calcular o espaço médio necessário (neste ambiente de testes) a cada segundo pelos algoritmos.

Para o algoritmo SPHINCS foi realizado teste de assinatura. Para realizar este teste, foram definidas três mensagens de tamanhos diferentes, sendo de 1KB, 100KB, e 1MB. Nos gráficos cada tamanho de mensagem é devidamente indicado através da legenda. Para gerar estas mensagens, foi utilizado o comando *dd* no Linux, detalhado a seguir. Para este algoritmo também foi realizado teste de verificação da mensagem.

```
$ dd if=/dev/zero of=mensagem_1KB bs=1K count=1
$ dd if=/dev/zero of=mensagem_100KB bs=100K count=1
$ dd if=/dev/zero of=mensagem_1MB bs=1M count=1
```

3.1.1 RSA

O teste com o RSA teve como objetivo medir a velocidade média de geração de chaves, tempo médio de assinaturas e verificações. Tais dados podem e serão comparados com os algoritmos que estão no processo de padronização do NIST. Para realizar o teste, foi utilizada a biblioteca de desempenho do *openssl* utilizando o seguinte comando no terminal:

```
$ openssl speed rsa
```

Esse comando gera por padrão chaves de 512, 1024, 2048, 4096, 7680 e 15360 bits. O teste foi realizado 10 vezes com o objetivo de obter um tempo médio. O teste com o openssl retorna o número de chaves públicas e privadas geradas em 10 segundos, possibilitando o cálculo de quanto tempo tomou para gerar cada par de chaves.

Na figura 3 observa-se que, para chaves a partir 2048 bits, a curva de crescimento do tempo se acentua mais.

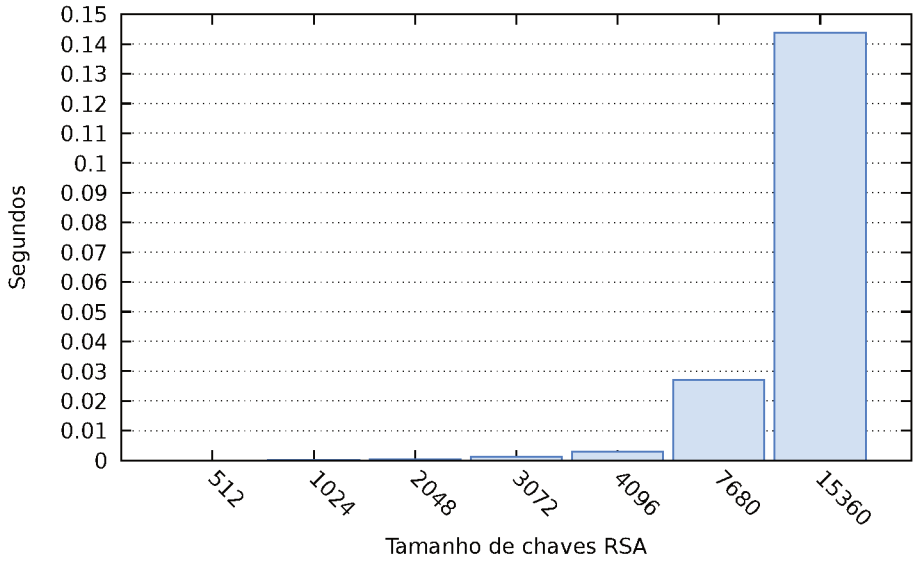


Figura 3 – Tempo médio de geração de chaves RSA

Pode-se observar o número de chaves geradas, em média, a cada segundo pelo RSA na Figura 4. Pela figura, podemos notar que para o RSA com chaves de 15360 bits, foram geradas cerca de 6,95 chaves por segundo.

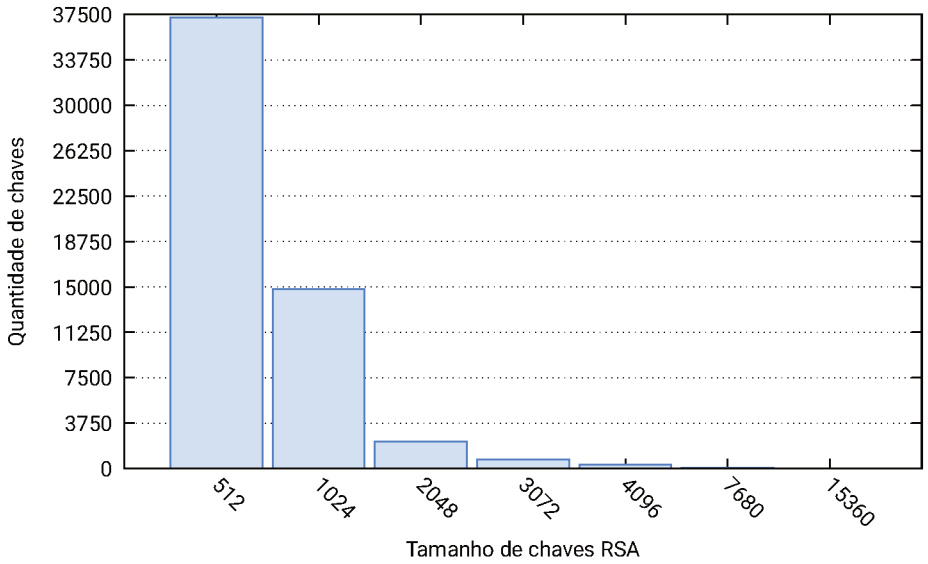


Figura 4 – Quantidade média de chaves RSA geradas a cada segundo

Assim como na geração de chaves, podemos observar no desempenho do RSA que, com chaves a partir de 4096 bits, tem um aumento exponencial na relação entre o tempo para realizar as assinaturas e as verificações e o tamanho das chaves. Tais gráficos podem ser observados nas Figuras 5 e 6 respectivamente.

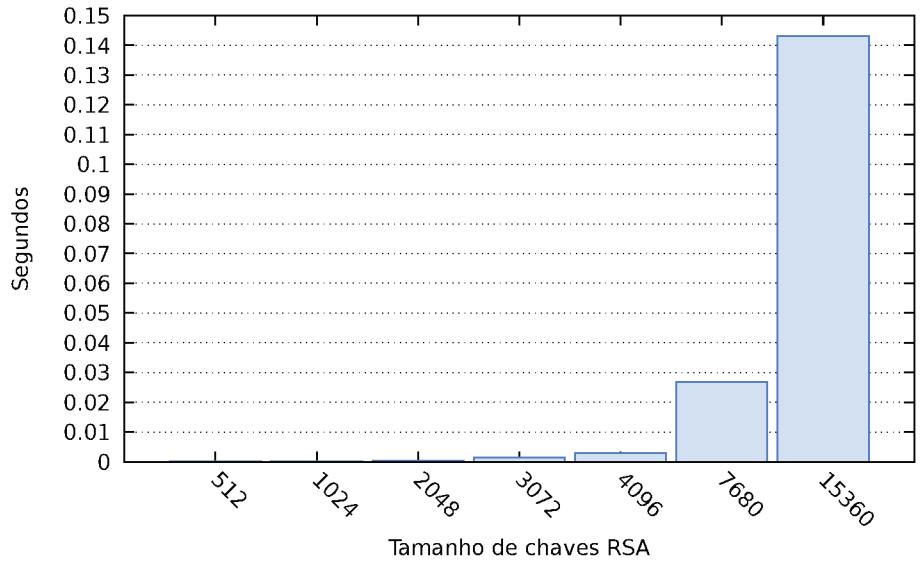


Figura 5 – Tempo médio de assinatura RSA

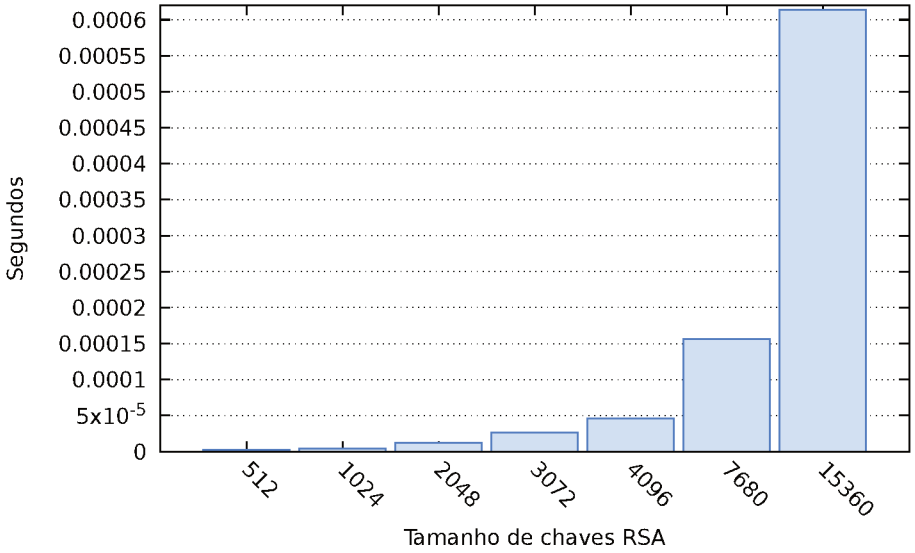


Figura 6 – Tempo médio para verificar uma assinatura RSA

3.1.2 McEliece

Após baixar o código do algoritmo de McEliece, é necessário compilá-lo para gerar o arquivo *run* e então executá-lo. Instruções de como compilar esse algoritmo são apresentadas no apêndice B.

As versões disponíveis do algoritmo se encontram na tabela abaixo. Na documentação presente em (ALBRECHT *et al.*, 2020) estão disponíveis informações sobre as variantes do algoritmo. Por exemplo, as versões que terminam em "f" são mais rápidas e possuem diferentes algoritmos para geração de chave, mas algoritmos idênticos para encriptação e decriptação.

No Quadro 8 temos o enquadramento de cada versão do algoritmo nas categorias de segurança do NIST, conforme explicitadas na Seção 2.2.1.

Quadro 8 – Implementações do McEliece disponíveis e categoria de segurança do NIST

Implementação	**Categoria**
mceliece348864	1
mceliece348864f	1
mceliece460896	3
mceliece460896f	3
mceliece6688128	5
mceliece6688128f	5
mceliece6960119	5
mceliece6960119f	5
mceliece8192128	5
mceliece8192128f	5

Podemos observar no Quadro 9 e com base em (ALBRECHT *et al.*, 2020), os

Quadro 9 – Tamanhos das chaves McEliece pública e privada em Bytes

Implementação	**Chave Pública**	**Chave Privada**
348864	261.120	6.492
348864f	261.120	6.492
460896	524.160	13.608
460896f	524.160	13.608
6688128	1.044.992	13.932
6688128f	1.044.992	13.932
6960119	1.047.319	13.948
6960119f	1.047.319	13.948
8192128	1.357.824	14.120
8192128f	1.357.824	14.120

primeiros 4 dígitos indicam o tamanho do código e os últimos dígitos representam a capacidade garantida de corrigir erros. Com isso, podemos observar que o tamanho do código tem influência sobre o tamanho da chave pública e privada. Ao aumentar o tamanho do código, o tamanho das chaves cresce. A chave pública é fortemente impactada com isso, levando esta para mais de 1MB.

Para a geração de chaves, podemos observar na Figura 7 que as versões "f" do algoritmo são mais rápidas. Com base nesta figura, cujos dados estão disponíveis no Apêndice A, através do Quadro 14, podemos calcular qual é o número média de chaves criadas a cada segundo, mostrado no Quadro 8.

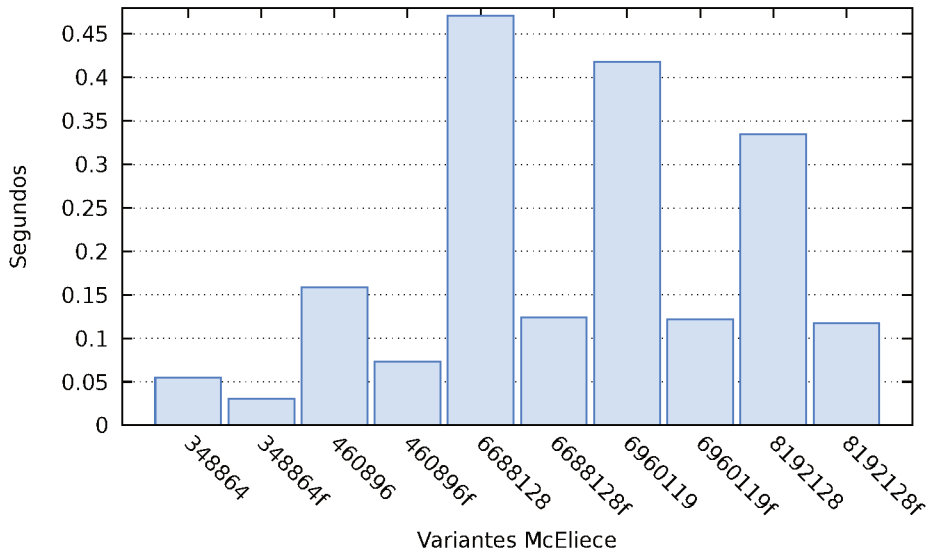


Figura 7 – Tempo médio de geração de chaves McEliece (em segundos)

A partir da Figura 8, podemos calcular qual seria, em média, o espaço requerido em armazenamento a cada segundo, conforme mostrado na Figura 9.

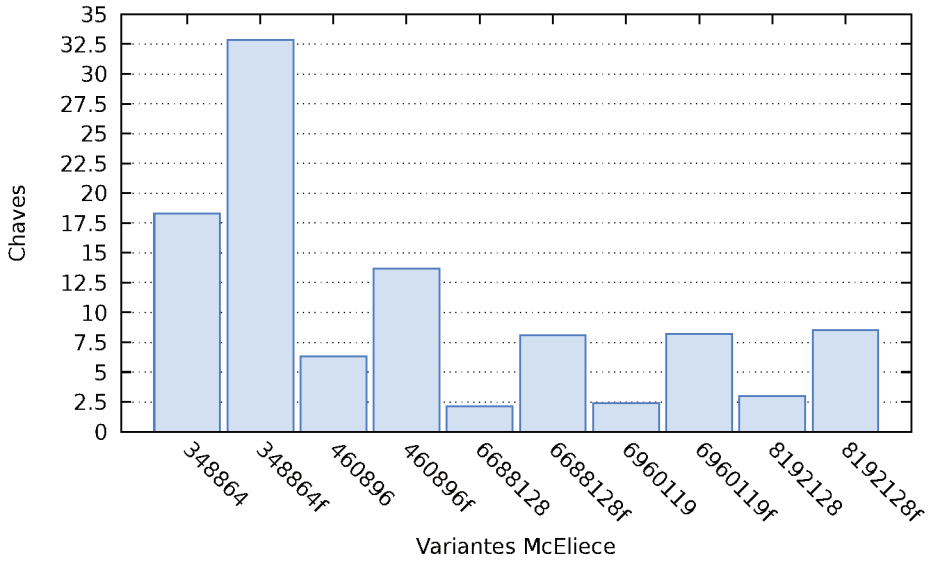


Figura 8 – Número médio de chaves McEliece geradas (por segundo)

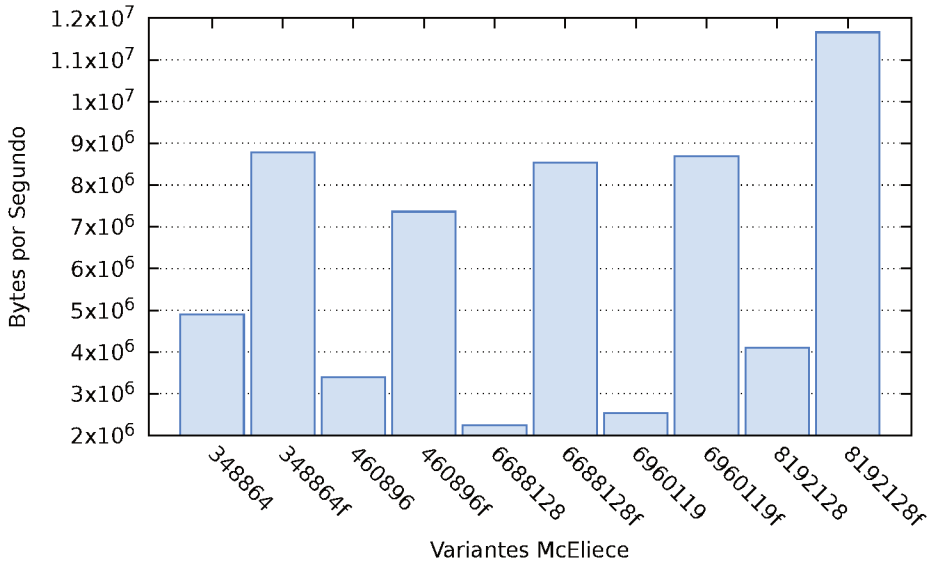


Figura 9 – Espaço médio utilizado pelas chaves McEliece geradas a cada segundo em Bytes

Observando a Figura 9 podemos ter uma ideia do impacto gerado pelo tamanho da chave pública gerada pela implementação mceliece8192128f necessitando, neste ambiente de testes, pouco mais de 11MB a cada segundo para armazenar as chaves geradas.

3.1.3 SPHINCS

No código do SPHINCS, há várias implementações do algoritmo e também um benchmark. Na documentação encontrada em (AUMASSON *et al.*, 2020), os autores afirmam que foram adicionadas para a segunda rodada instâncias simples (marcadas com “simple”) e robustas (marcadas com “robust”). As instâncias simples tem a

vantagem de serem mais rápidas, porém possuem como desvantagem o argumento da segurança que em sua totalidade só se aplica no modelo oráculo aleatório. Já as instâncias robustas possuem um argumento de segurança mais conservador, porém são mais lento.

Os autores, consideraram três maneiras diferentes de implementar as famílias de funções criptográficas como diferentes sistemas de assinaturas. Os diferentes sistemas de assinaturas são SHAKE256, SHA-256 e Haraka, onde, para este último algoritmo, afirmam que não é uma função de hash aprovada pelo NIST. Há também um mecanismo onde é possível efetuar trocas entre tamanho e velocidade. Algoritmos terminados em “s” representam tamanho de assinatura menor e terminados em “f” possuem maior velocidade do algoritmo. Por exemplo, SPHINCS Haraka 256s opta pelo mecanismo com menor tamanho de assinatura enquanto que SPHINCS Haraka 256f opta por maior velocidade.

Instruções de como baixar e compilar o algoritmo estão disponíveis no Apêndice B.

No Quadro 10 temos o enquadramento de cada versão do algoritmo nas categorias de segurança do NIST, conforme explicitadas na Seção 2.2.1.

Quadro 10 – Categoria de segurança do NIST para o SPHINCS

Implementação	**Categoria**
sphincs-128s	1
sphincs-128f	1
sphincs-192s	3
sphincs-192f	3
sphincs-256s	5
sphincs-256f	5

Como observado nos benchmarks, o esquema possui uma assinatura de 29,98 KiB, uma chave pública de 0,06 KiB, uma chave privada de 0,12 KiB. No Quadro 11, podemos ver os tamanhos das chaves e da assinatura para cada variante do algoritmo.

Quadro 11 – Tamanhos de Chaves Públicas, Chaves Privadas e Assinaturas do SPHINCS em Bytes

Implementação	**Chave Pública**	**Chave Privada**	**Assinatura**
sphincs-128s	32	64	7.856
sphincs-128f	32	64	17.088
sphincs-192s	48	96	16.224
sphincs-192f	48	96	35.664
sphincs-256s	64	128	29.792
sphincs-256f	64	128	49.856

Nas Seções seguintes, temos a Geração de Chave, Assinatura e Verificação deste algoritmo com cada uma de suas variantes.

3.1.3.1 Geração de Chaves

Primeiramente, ao observar os gráfico da Figura 10, já podemos notar que as versões “f” são de fato mais rápidas do que suas respectivas versões “s”. Além disso, comparando cada implementação “f” para Haraka, SHA256 e SHAKE256, o desvio padrão é menor quando comparado com as versões “s”. Ao utilizar o SHA-256, a geração de chaves teve um aumento no desempenho. Essa melhoria de desempenho é vista mais nitidamente nas versões “s” do algoritmo.

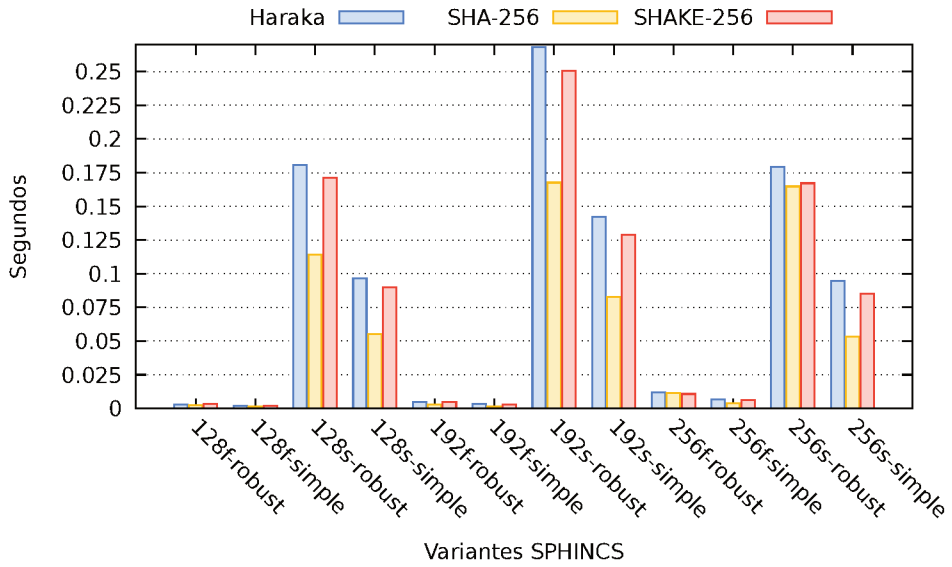


Figura 10 – Tempo médio de geração de chaves SPHINCS Haraka, SHA-256 e SHAKE-256 (em segundos)

Com base no tempo médio de geração de chaves, foi possível calcular o número médio de chaves geradas a cada segundo, ilustrado na Figura 11. Nesta figura, podemos ver que o SPHINCS-SHA-256-128f-simple é a implementação que gera mais chaves a cada segundo.

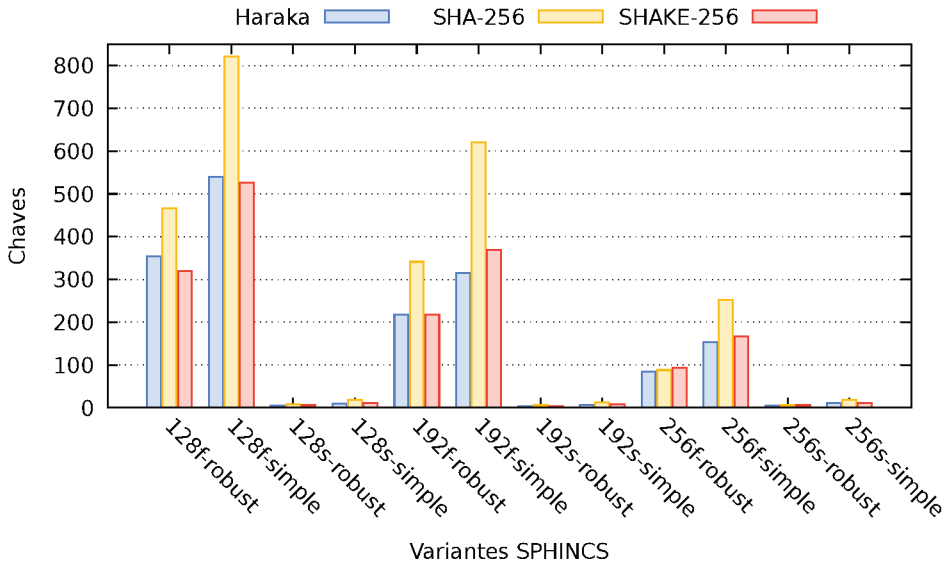


Figura 11 – Número médio de chaves SPHINCS geradas por segundo

Com base nos dados da Figura 11 foi possível calcular o espaço médio total utilizado pela soma do espaço da chave pública e privada para cada implementação, ilustrado na Figura 12.

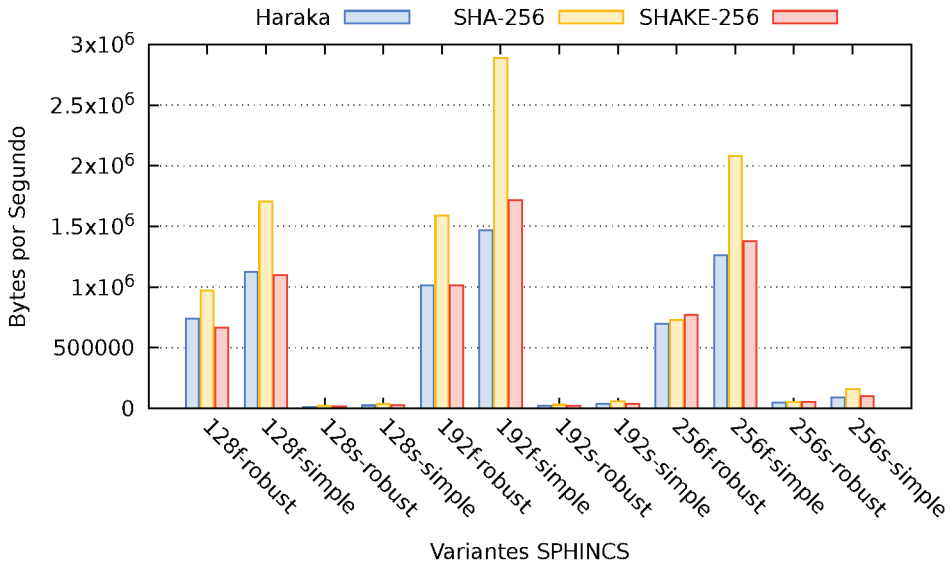


Figura 12 – Espaço utilizado em média pelas chaves pública e privada do SPHINCS a cada segundo em Bytes

Observamos, na Figura 12, que as versões “f” necessitam de maior espaço quando comparado com as versões “s” de seus respectivos tamanhos. Nesta figura também podemos notar que o algoritmo SPHINCS-SHA-256-192f-simple é o que necessita de maior espaço a cada segundo.

3.1.3.2 Assinatura

Para as assinaturas, visando uma melhor visualização, foram separados um gráfico por algoritmo (Haraka, SHA-256, SHAKE-256). Em cada um destes gráficos, temos três tamanhos diferentes de mensagens a serem assinadas (1KB, 100KB e 1MB). A forma como estas mensagens foram criadas já foi explicada anteriormente. Para facilitar a visualização, a escala dos gráficos relacionados ao tempo médio de assinatura foi mantida.

Nas Figuras 13, 14 e 15, podemos observar que as variantes “f” obtiveram melhor desempenho do que as variantes “s” e as variantes “simple” obtiveram melhor desempenho do que as variantes “robust”. Ao utilizar o SHA-256, apresentada na Figura 14, o algoritmo foi mais rápido quando comparado aos demais, Haraka e Shake-256.

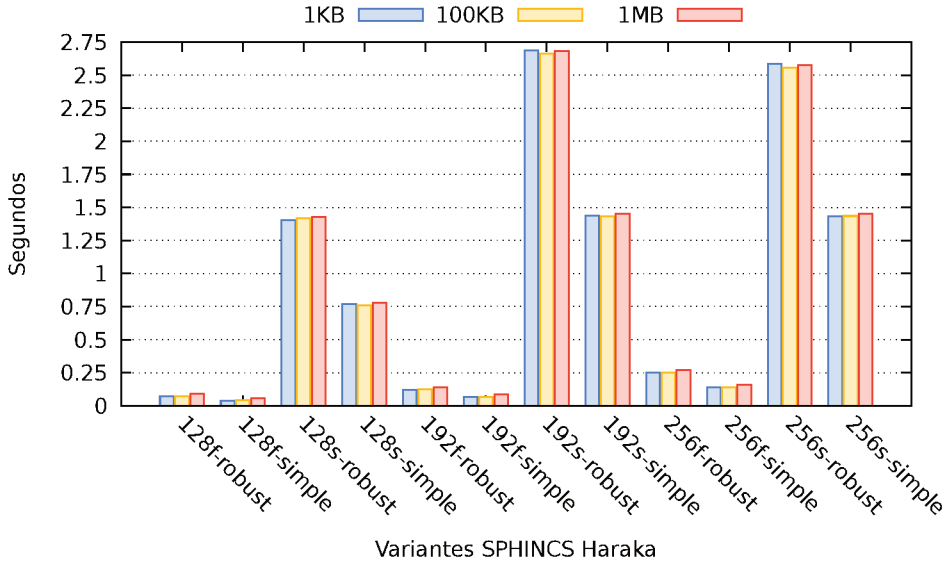


Figura 13 – Tempo médio de assinatura SPHINCS Haraka (em segundos)

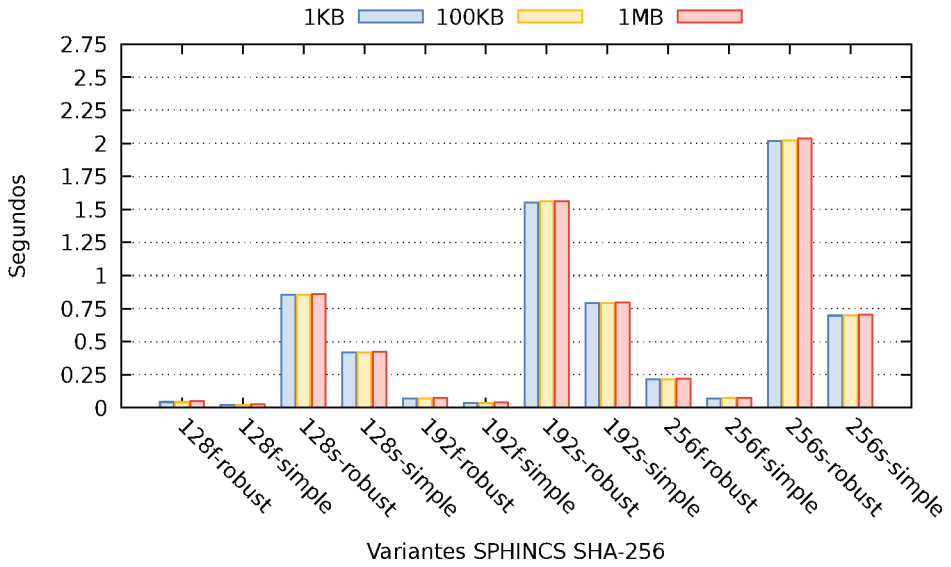


Figura 14 – Tempo médio de assinatura SPHINCS SHA-256 (em segundos)

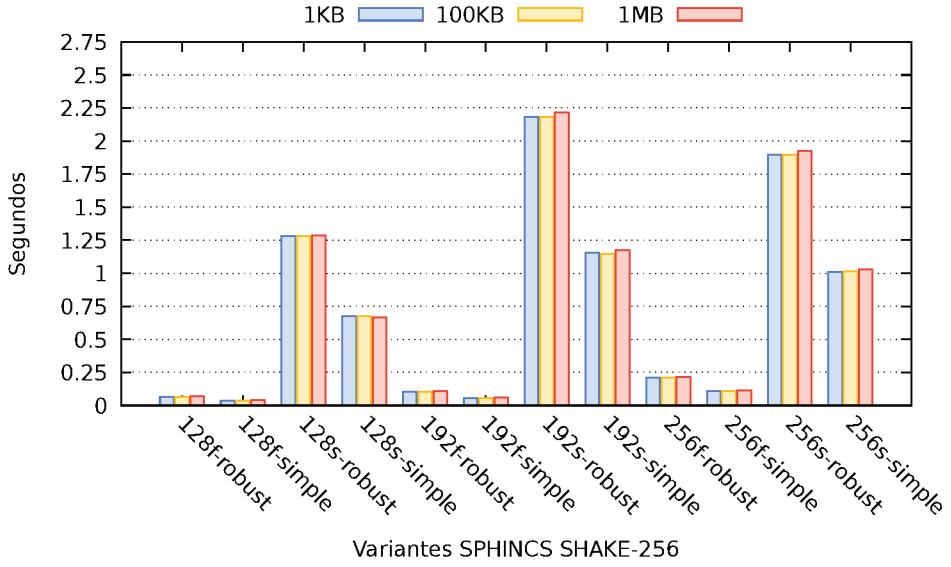


Figura 15 – Tempo médio de assinatura SPHINCS SHAKE-256 (em segundos)

Baseado nestes dados, foi possível calcular uma média de assinaturas geradas a cada segundo. Com isso observa-se que as variantes “f” geraram mais chaves, em média, quando comparadas com as variantes “s”. Isso ocorre como consequência da variante “f” ser mais rápida.

É importante destacar que a escala destes gráficos não é a mesma. Tais dados sobre número de chaves geradas estão disponíveis das Figuras 16, 17 e 18.

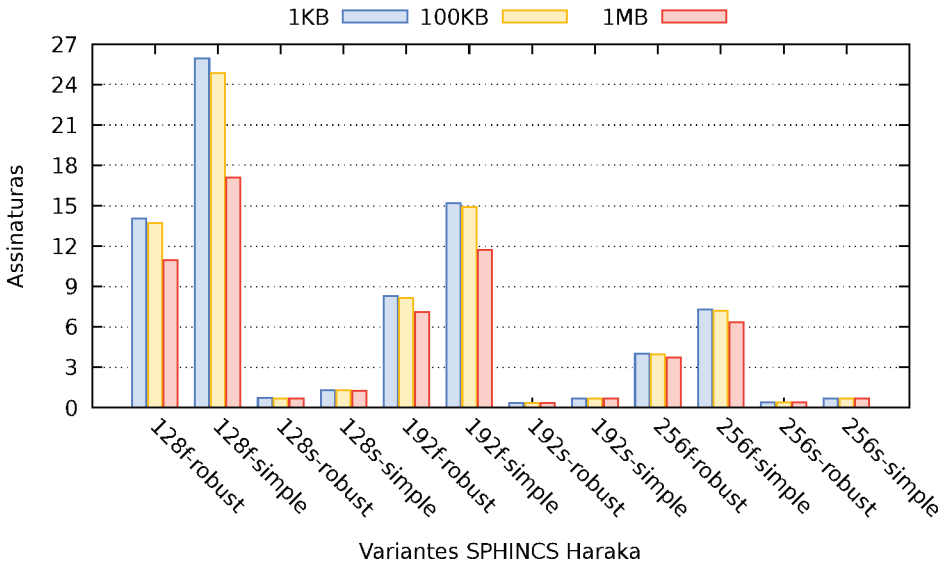


Figura 16 – Numero médio de assinaturas geradas pelo SPHINCS Haraka a cada segundo

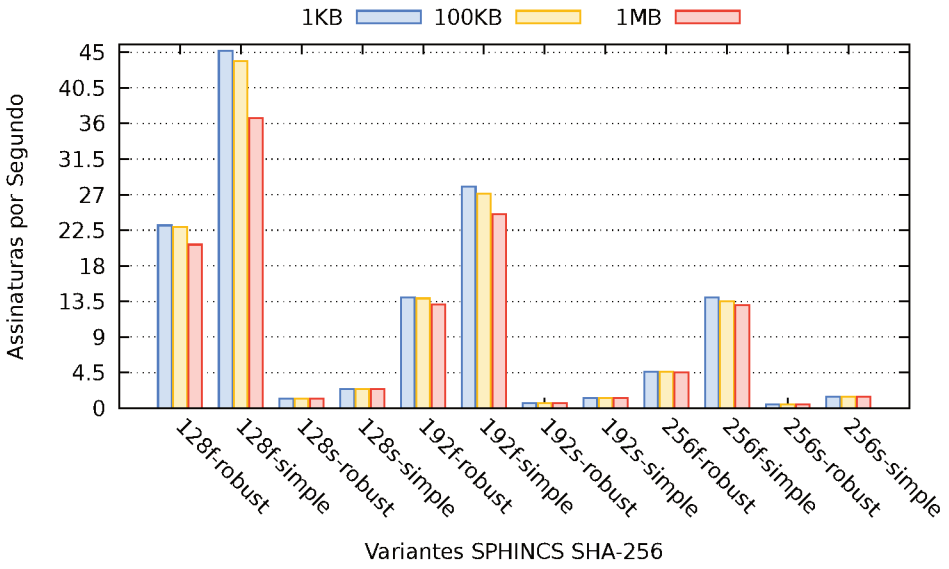


Figura 17 – Número médio de assinaturas geradas pelo SPHINCS SHA-256 a cada segundo

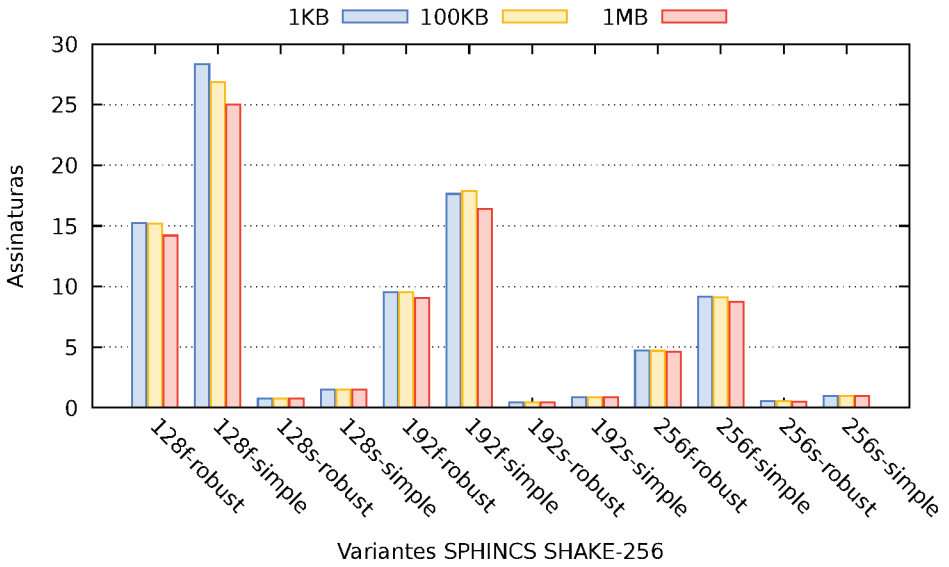


Figura 18 – Número médio de assinaturas geradas pelo SPHINCS SHAKE-256 a cada segundo

Com base nestes dados de quantidade de chaves geradas a cada segundo, foi possível calcular o espaço médio necessário a cada segundo para a armazenar somente a assinatura, não considerando o tamanho da mensagem ou arquivo assinado.

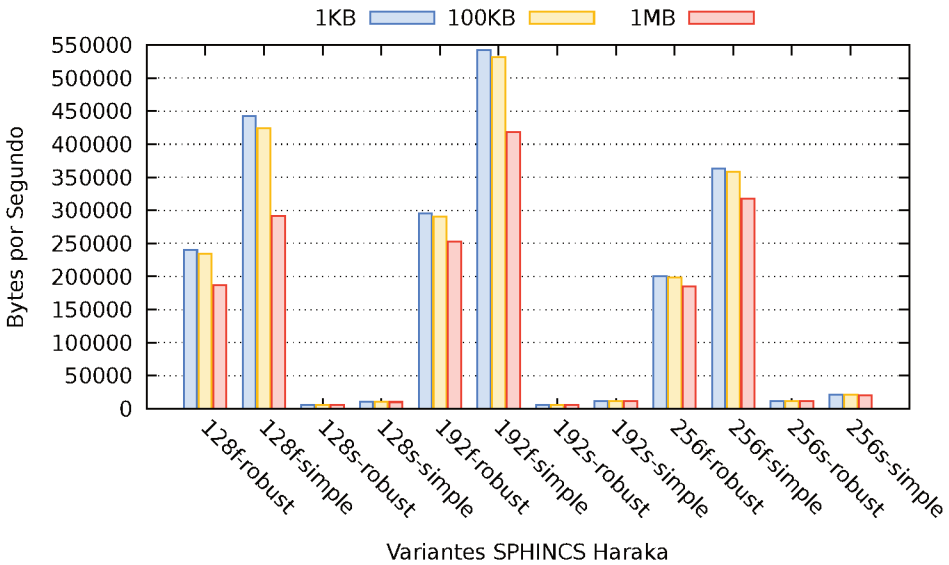


Figura 19 – Espaço médio utilizado pela geração de assinaturas SPHINCS Haraka a cada segundo em Bytes

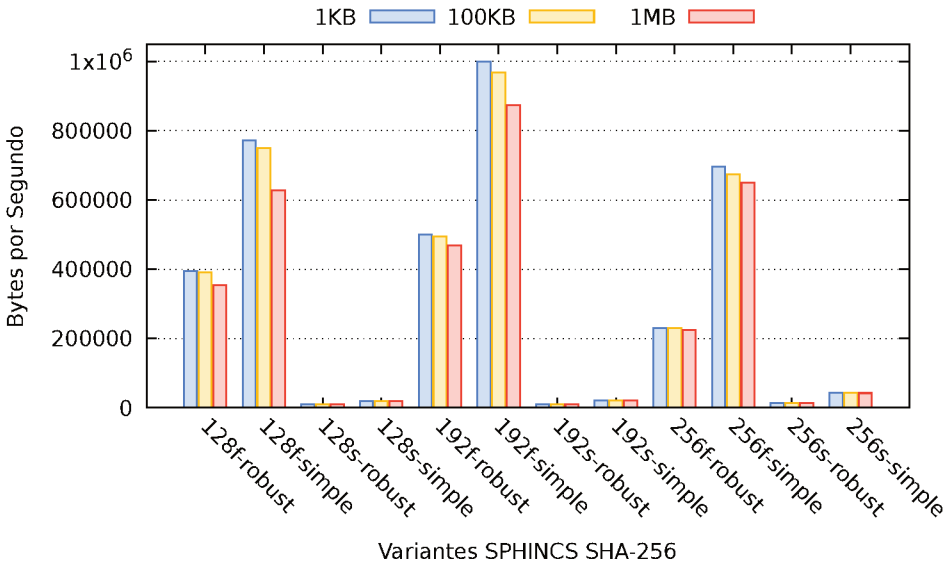


Figura 20 – Espaço médio utilizado pela geração de assinaturas SPHINCS SHA-256 a cada segundo em Bytes

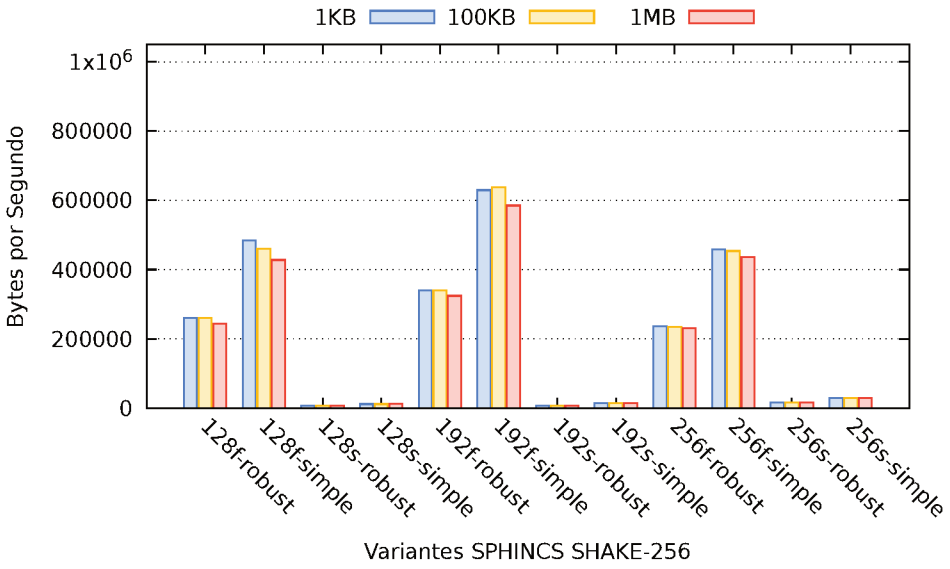


Figura 21 – Espaço médio utilizado pela geração de assinaturas SPHINCS SHAKE-256 a cada segundo em Bytes

3.1.3.3 Verificação

Para a verificação, também separamos as variantes Haraka, SHA-256 e SHAKE-256 em gráficos separados mantendo a escala para melhor visualização e comparação. Os gráficos contém a legenda referente aos tamanhos de mensagens.

Podemos notar que na variante Haraka, apresentada pela Figura 22, a diferença em relação a verificação de uma mensagem de 1MB em relação a de 100KB é muito maior quando comparado às mensagens de 100KB em relação às de 1KB, mostrando que o tempo de verificação de mensagem não é linear ao tamanho da mensagem para

a variante Haraka. Já nas variantes SHA-256 e SHAKE-256, representadas respectivamente nas Figuras 24 e 23 o tempo de verificação de mensagem de 1MB em relação aos outros tamanhos (100KB e 1KB) não cresce tanto como na variante Haraka.

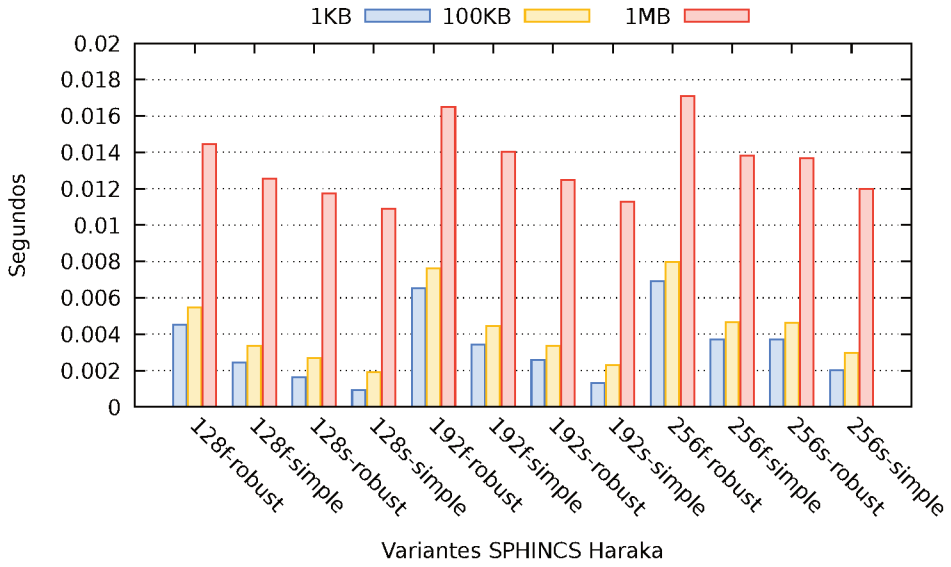


Figura 22 – Tempo médio de verificação SPHINCS Haraka (em segundos)

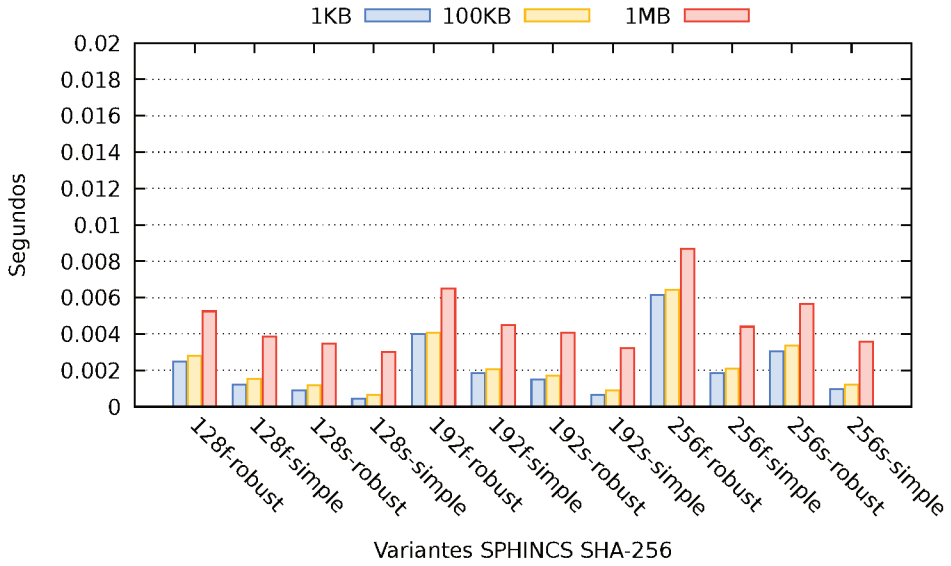


Figura 23 – Tempo médio de verificação SPHINCS SHA-256 (em segundos)

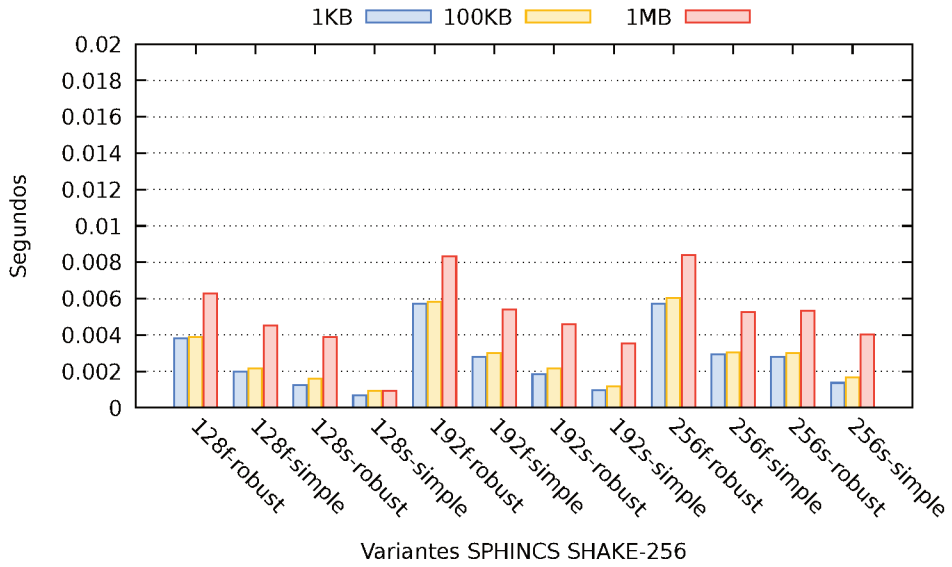


Figura 24 – Tempo médio de verificação SPHINCS SHAKE-256 (em segundos)

Apesar dos algoritmos de verificação das variantes SHA-256 e SHAKE-256 apresentarem desempenho parecidos, o SHA-256, apresentado na Figura 23, foi levemente mais rápido.

3.1.4 NTRU

O código fonte contendo as variantes do algoritmo NTRU foram baixadas do site do NIST (NIST, 2017d) e compiladas para a execução dos testes. Instruções mais detalhadas a respeito do processo de compilação estão presentes no Apêndice B.

No Quadro 12 temos o enquadramento de cada variação do algoritmo nas categorias de segurança do NIST, conforme explicitadas na Seção 2.2.1. Os autores propõem quatro implementações que podem ser vistas através deste mesmo quadro.

Quadro 12 – Categoria de segurança do NIST para o NTRU

Implementação	**Categoria**
ntruhps2048509	1
ntruhps2048677	3
ntruhrss701	3
ntruhps4096821	5

Podemos observar no Quadro 13 as variantes do NTRU ordenadas por nível de segurança. Já na Figura 25 observa-se perda de desempenho quando o nível de segurança é aumentado, com o aumento no tamanho das chaves pública e privada.

Quadro 13 – Tamanho das chaves geradas pelo NTRU em Bytes

Implementação	**Chave Pública**	**Chave Privada**
ntruhps2048509	699	935
ntruhps2048677	930	1234
ntruhrss701	1138	1450
ntruhps4096821	1230	1590

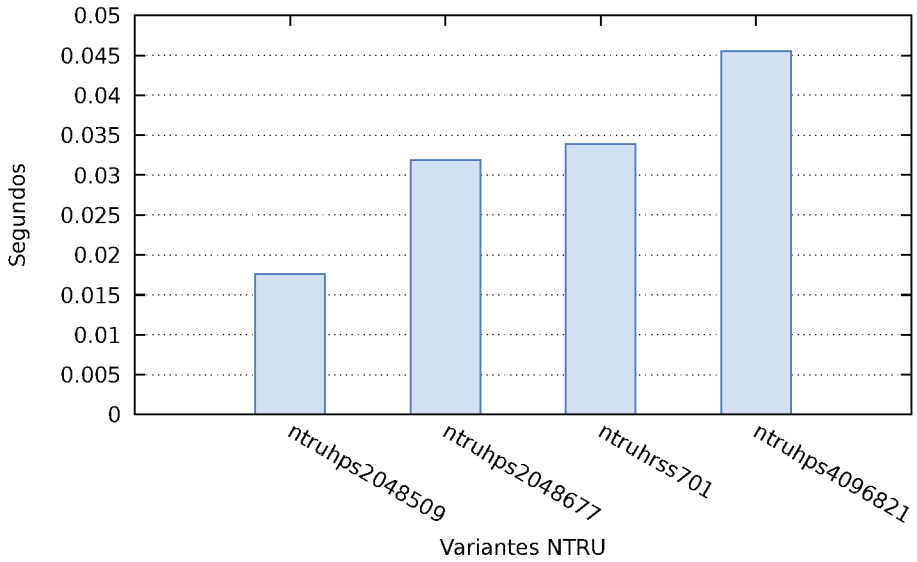


Figura 25 – Tempo médio de geração de Chaves NTRU (em segundos)

A quantidade média de chaves geradas, atinge seu pico com aproximadamente 55 chaves a cada segundo na versão mais rápida e aproximadamente 20 chaves a cada segundo na versão mais lenta, porém, mais segura.

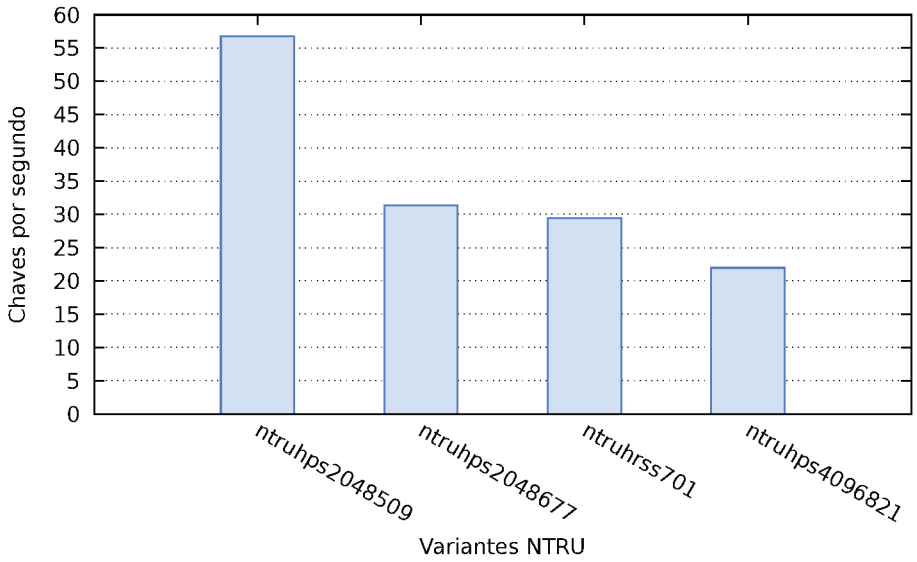


Figura 26 – Número médio de chaves NTRU geradas a cada segundo

A partir dos dados do Quadro 13 e Figura 25 foi possível calcular o espaço médio utilizado para armazenar as chaves a cada segundo. Na versão mais rápida do algoritmo, chega-se a um pico de aproximadamente 90KB a cada segundo, e cerca de 60KB a cada segundo para a versão mais lenta. É importante destacar que a versão mais lenta gera menos chaves, porém, as chaves são maiores.

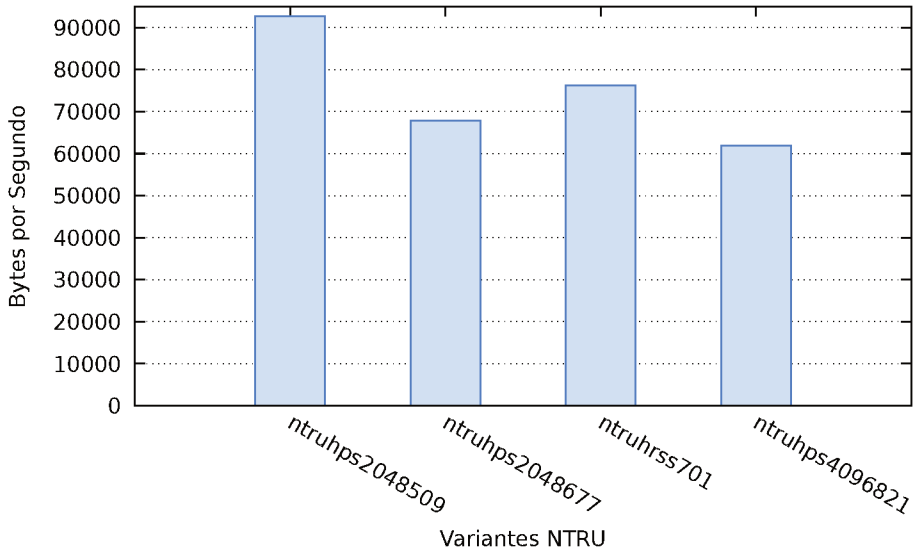


Figura 27 – Espaço utilizado em média pelas chaves NTRU geradas a cada segundo em Bytes

3.1.5 Rainbow

Apesar do algoritmo Rainbow ser promissor para assinaturas e constar na terceira rodada do NIST, o mesmo foi quebrado recentemente (BEULLENS, 2022). Em função disso, o Rainbow foi exceptuado destes testes.

3.2 COMPARATIVO ENTRE O RSA E AS TÉCNICAS PÓS-QUÂNTICAS

O algoritmo entre as técnicas pós-quânticas que teve melhor desempenho na geração de chaves foi o SPHINCS com a variante sphincs-sha256-256f-simple que obteve o tempo médio de 0,003967 segundos. Esse desempenho é semelhante ao RSA com chave de 4096 bits, onde o tempo médio para geração de chaves foi de 0,003022 segundos.

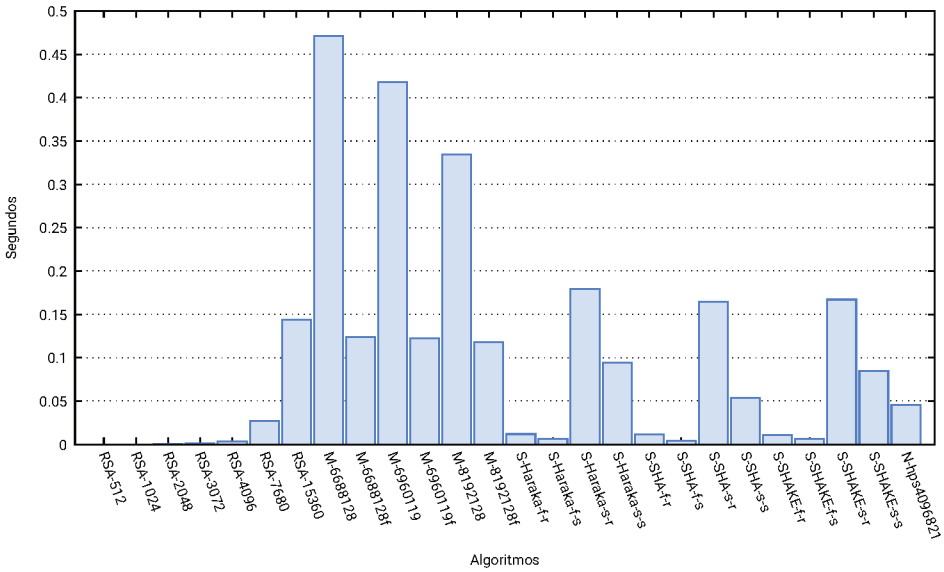


Figura 28 – Geração de chaves

Para a assinatura, podemos comparar o RSA com o SPHINCS. Entre as variações de algoritmos pós-quânticos apresentados, a que obteve melhor desempenho foi a variante sphincs-sha256-128f-simple, que levou em média 0,022137 segundos para assinar uma mensagem de 1KB, em média 0,022782 segundos para assinar uma mensagem de 100KB e em média 0,027242 segundos para assinar uma mensagem de 1MB. O algoritmo RSA foi melhor em desempenho até o RSA com chave de 4096 bits, onde levou em média 0,0030223 segundos. Para as variantes do RSA com chave maior ou igual que 7680 bits, a variante citada do SPHINCS obteve desempenho melhor.

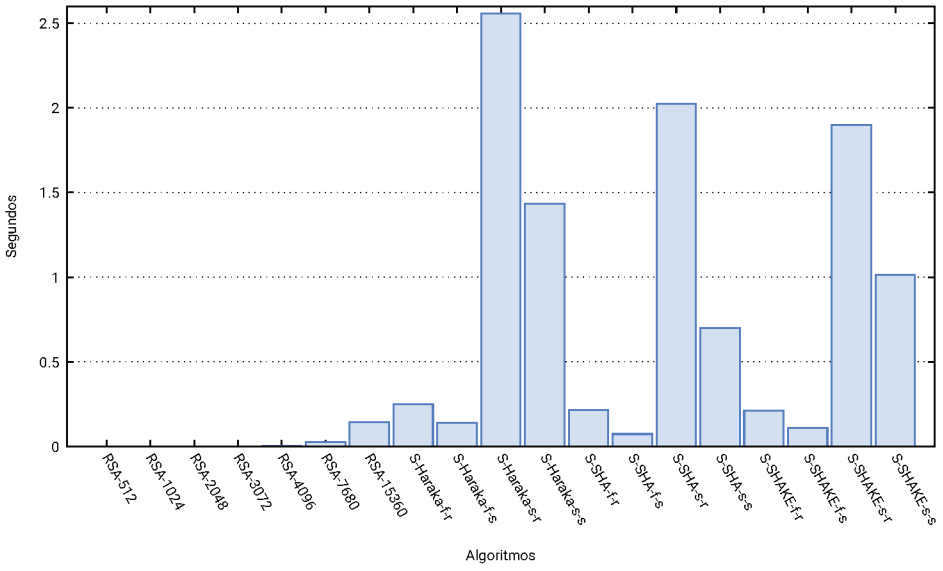


Figura 29 – Assinatura

No caso da verificação, o algoritmo que obteve melhor desempenho foi o SPHINCS com a variante sphincs-sha256-128s-simple. O algoritmo levou em média

0,00075969 segundos para verificar uma mensagem de 1KB, 0,00118959 segundos para mensagem de 100KB e 0,00570408 segundos para mensagem de 1MB. Em comparação com a variante do SPHINCS anteriormente citada, o RSA obteve melhor desempenho.

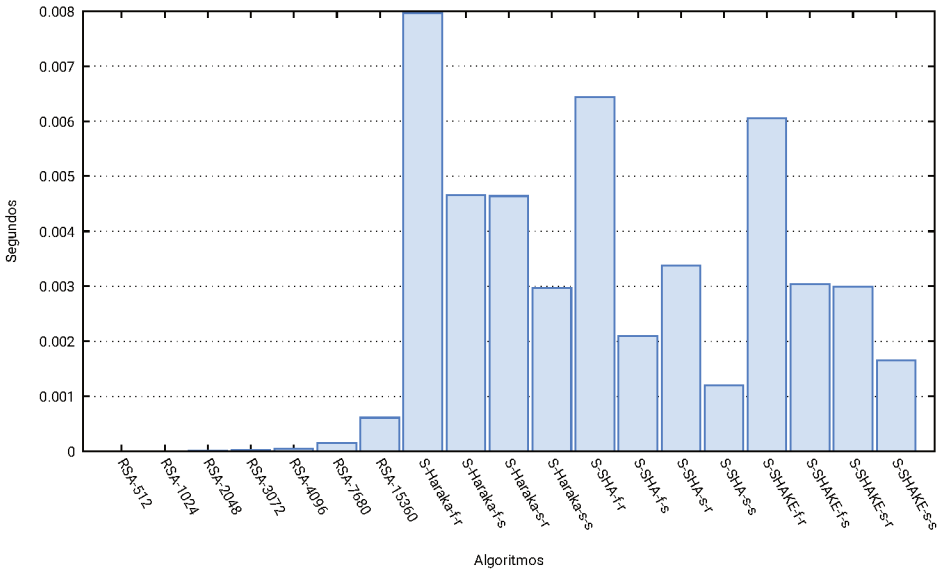


Figura 30 – Verificação

Esta comparação entre as técnicas pós-quânticas e o RSA é importante para se ter uma ideia do quanto pior pode ser um algoritmo pós-quântico em termos de tempo de execução. Lembrando que esses algoritmos devem ser executados no mesmo hardware que executamos o RSA hoje.

4 CONCLUSÕES E CONSIDERAÇÕES FINAIS

O processo de padronização do NIST tem sido muito importante para coletar algoritmos que prometem ser resistentes a computadores quânticos. Neste processo, se observa que houve melhorias nos algoritmos e espera-se que isso aconteça também na medida em que estes algoritmos passem para as fases posteriores.

Este trabalho buscou fazer um levantamento acerca das principais famílias de técnicas pós-quânticas em pesquisa no momento, apresentando as ideias gerais sobre cada técnica e mostrando um conjunto de testes das mesmas.

O conjunto de testes foi importante para analisar o desempenho dos novos algoritmos candidatos à padronização e suas variações. Embora a implementação desses algoritmos não seja ótima, já serviu de parâmetro para comparação com os algoritmos usados atualmente.

A evolução das etapas futuras do processo de padronização poderia se tornar um trabalho futuro. O levantamento das melhorias dos algoritmos em cada fase de padronização também poderia fazer parte de um trabalho futuro bem como o levantamento das melhorias realizadas nos algoritmos e qual tem se monstrado melhor. Outra possibilidade para trabalho futuro seria escolher uma técnica e desenvolver um estudo de análise de sua complexidade em termos de cada processo, por exemplo, geração de chave, criptografar, descriptografar, assinar, verificação de assinatura, etc. Ademais, poderia ser feito uma análise sobre a complexidade computacional de uma técnica específica, verificando se o problema matemático que a técnica usa é comprovadamente um problema NP-Completo.

REFERÊNCIAS

AJTAI, Miklós; DWORK, Cynthia. A Public-Key Cryptosystem with Worst-Case/Average-Case Equivalence. *In:* PROCEEDINGS of the Twenty-Ninth Annual ACM Symposium on Theory of Computing. El Paso, Texas, USA: Association for Computing Machinery, 1997. (STOC '97), p. 284–293. DOI: 10.1145/258533.258604. Disponível em: <%5Curl%7Bhttps://doi.org/10.1145/258533.258604%7D>. Citado na p. 45.

ALBRECHT, Martin R. *et al.* **Classic McEliece: conservative code-based cryptography**. [*S.l.*: *s.n.*], 2020. Disponível em: <https://classic.mceliece.org/nist/mceliece-20201010.pdf>. Acesso em: 5 ago. 2020. Citado na p. 57.

AUMASSON, Jean-Philippe *et al.* **SPHINCS+ – Submission to the 3rd round of the NIST post-quantum project**. Inglês. NIST. Out. 2020. Disponível em: <https://sphincs.org/data/sphincs+-round3-specification.pdf>. Acesso em: 5 ago. 2020. Citado nas pp. 35, 59.

BALAMURUGAN, Chithralekha *et al.* **Code-based Post-Quantum Cryptography**. Abr. 2021. DOI: 10.20944/preprints202104.0734.v1. Disponível em: <https://www.preprints.org/manuscript/202104.0734/v1>. Citado nas pp. 15, 53.

BERNSTEIN, Daniel J.; BUCHMANN, Johannes; DAHMEN, Erik (Ed.). **Post-Quantum Cryptography**. [*S.l.*]: Springer Berlin Heidelberg, 2009. DOI: 10.1007/978-3-540-88702-7. Disponível em: <https://doi.org/10.1007/978-3-540-88702-7>. Citado nas pp. 15, 20, 23, 32.

BERNSTEIN, Daniel J.; HOPWOOD, Daira *et al.* SPHINCS: Practical Stateless Hash-Based Signatures. *In:* OSWALD, Elisabeth; FISCHLIN, Marc (Ed.). **Advances in Cryptology – EUROCRYPT 2015**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015. P. 368–397. Citado nas pp. 35–40, 42.

BEULLENS, Ward. Breaking Rainbow Takes a Weekend on a Laptop. **Cryptology ePrint Archive**, 2022. Citado nas pp. 54, 71.

BILLET, Olivier; GILBERT, Henri. Cryptanalysis of Rainbow. *In:* DE PRISCO, Roberto; YUNG, Moti (Ed.). **Security and Cryptography for Networks**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. P. 336–347. Citado na p. 49.

BUCERZAN, Dominic; VLAD, Dragoi; TALÉ KALACHI, Hervé. Evolution of the McEliece Public Key Encryption Scheme. *In:* p. 129–149. DOI: 10.1007/978-3-319-69284-5_10. Citado nas pp. 26–28.

BUCHMANN, Johannes; DAHMEN, Erik; SZYDLO, Michael. Hash-based Digital Signature Schemes. *In:* **Post-Quantum Cryptography**. Edição: Daniel J. Bernstein, Johannes Buchmann e Erik Dahmen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. P. 35–93. ISBN 978-3-540-88702-7. DOI: 10.1007/978-3-540-88702-7_3. Disponível em: <https://doi.org/10.1007/978-3-540-88702-7_3>. Citado nas pp. 32–35, 37, 38.

BUTIN, D. Hash-Based Signatures: State of Play. **IEEE Security Privacy**, v. 15, n. 4, p. 37–43, 2017. ISSN 1540-7993. DOI: 10.1109/MSP.2017.3151334. Citado nas pp. 32, 33, 36, 43, 48.

CHEN, Jiahui *et al.* Identity-Based Signature Schemes for Multivariate Public Key Cryptosystems. **The Computer Journal**, v. 62, n. 8, p. 1132–1147, mar. 2019. ISSN 0010-4620. DOI: 10.1093/comjnl/bxz013. eprint: http://oup.prod.sis.lan/comjnl/article-pdf/62/8/1132/29162306/bxz013.pdf. Disponível em: <https://doi.org/10.1093/comjnl/bxz013>. Citado nas pp. 48, 49.

CHEN, Lily *et al.* **Report on Post-Quantum Cryptography**. [*S.l.*]: National Institute of Standards e Technology(NIST), 2016. Citado nas pp. 15, 17, 19, 20, 23, 32, 48.

CHEROWITZO, Bill. **Intro to Code Theory**. Inglês. Disponível em: <http://www-math.ucdenver.edu/~wcherowi/courses/m5410/codingintro.pdf>. Acesso em: 2 out. 2020. Citado na p. 23.

CHEROWITZO, Bill. **The McEliece Cryptosystem**. Inglês. Disponível em: <http://www-math.ucdenver.edu/~wcherowi/courses/m5410/ctcmcel.html>. Acesso em: 18 set. 2020. Citado na p. 30.

DING, J.; PETZOLDT, A. Current State of Multivariate Cryptography. **IEEE Security Privacy**, v. 15, n. 4, p. 28–36, 2017. DOI: 10.1109/MSP.2017.3151328. Citado na p. 52.

DING, Jintai; SCHMIDT, Dieter. Rainbow, a New Multivariable Polynomial Signature Scheme. *In:* p. 164–175. DOI: 10.1007/11496137_12. Citado nas pp. 49–52.

DING, Jintai; YANG, Bo-Yin. Multivariate Public Key Cryptography. *In:* **Post-Quantum Cryptography**. Edição: Daniel J. Bernstein, Johannes Buchmann e Erik Dahmen.

Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. P. 193–241. ISBN 978-3-540-88702-7. DOI: 10.1007/978-3-540-88702-7_6. Disponível em: <https://doi.org/10.1007/978-3-540-88702-7_6>. Citado nas pp. 48, 49.

DRAGOI, Vlad *et al.* Survey on Cryptanalysis of Code-Based Cryptography: from Theoretical to Physical Attacks. *In:* 2018 7th International Conference on Computers Communications and Control (ICCCC). [*S.l.*: *s.n.*], 2018. Citado nas pp. 16, 20.

GISIN, Nicolas *et al.* Quantum cryptography. **Rev. Mod. Phys.**, American Physical Society, v. 74, p. 145–195, 1 mar. 2002. DOI: 10.1103/RevModPhys.74.145. Disponível em: <https://link.aps.org/doi/10.1103/RevModPhys.74.145>. Citado na p. 19.

GROOT BRUINDERINK, Leon; HÜLSING, Andreas. "Oops, I Did It Again" – Security of One-Time Signatures Under Two-Message Attacks. *In:* ADAMS, Carlisle; CAMENISCH, Jan (Ed.). **Selected Areas in Cryptography – SAC 2017**. Cham: Springer International Publishing, 2018. P. 299–322. Citado nas pp. 32, 33.

HOFFSTEIN, Jeffrey; PIPHER, Jill; SILVERMAN, Joseph H. NTRU: A ring-based public key cryptosystem. *In:* BUHLER, Joe P. (Ed.). **Algorithmic Number Theory**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998. P. 267–288. Citado nas pp. 46, 47.

HÜLSING, Andreas. **WOTS+ - Shorter Signatures for Hash-Based Signature Schemes**. Inglês. 2017. Disponível em: <https://eprint.iacr.org/2017/965.pdf>. Acesso em: 6 ago. 2020. Citado na p. 36.

HÜLSING, Andreas; RIJNEVELD, Joost; SCHWABE, Peter. ARMed SPHINCS. *In:* CHENG, Chen-Mou *et al.* (Ed.). **Public-Key Cryptography – PKC 2016**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016. P. 446–470. Citado na p. 35.

MANFERDELLI, John. **Cryptanalysis: Error Correcting Codes**. Inglês. Disponível em: <https://courses.cs.washington.edu/courses/cse599r/08au/LectureNotes/Crypto599class10.pdf>. Acesso em: 6 out. 2020. Citado na p. 29.

MCELIECE, Robert. **The Theory of Information and Coding**. 2. ed. [*S.l.*]: Cambridge University Press, 2002. (Encyclopedia of Mathematics and its Applications). DOI: 10.1017/CBO9780511606267. Citado na p. 23.

MCELLICE. **A Public-Key System Based on Algebraic Coding Theory**. [*S.l.*], 1978. Citado nas pp. 20, 26.

MERKLE, Ralph C. A Certified Digital Signature. *In:* ADVANCES in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings. [*S.l.*]: Springer, 1989. (Lecture Notes in Computer Science), p. 218–238. DOI: 10.1007/0-387-34805-0_21. Citado na p. 35.

MICCIANCIO, Daniele; REGEV, Oded. Lattice-based Cryptography. *In:* **Post-Quantum Cryptography**. Edição: Daniel J. Bernstein, Johannes Buchmann e Erik Dahmen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. P. 147–191. ISBN 978-3-540-88702-7. DOI: 10.1007/978-3-540-88702-7_5. Disponível em: <https://doi.org/10.1007/978-3-540-88702-7_5>. Citado na p. 44.

NEJATOLLAHI, Hamid *et al.* Post-Quantum Lattice-Based Cryptography Implementations: A Survey. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 51, n. 6, jan. 2019. ISSN 0360-0300. DOI: 10.1145/3292548. Disponível em: <https://doi.org/10.1145/3292548>. Citado na p. 43.

NIST. **Post-Quantum Cryptography Standardization**. Inglês. NIST. Jan. 2017. Disponível em: <https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization>. Acesso em: 21 dez. 2020. Citado na p. 20.

NIST. **PQC Standardization Process: Second Round Candidate Announcement**. Inglês. NIST. Jan. 2019. Disponível em: <https://csrc.nist.gov/news/2019/pqc-standardization-process-2nd-round-candidates>. Acesso em: 22 jul. 2020. Citado na p. 24.

NIST. **PQC Third Round Candidate Announcement**. [*S.l.*: *s.n.*], jul. 2020. Disponível em: <https://csrc.nist.gov/News/2020/pqc-third-round-candidate-announcement>. Citado na p. 21.

NIST. **Round 1 Submissions**. Inglês. NIST. Jan. 2017. Disponível em: <https://csrc.nist.gov/Projects/post-quantum-cryptography/Round-1-Submissions>. Acesso em: 15 jul. 2020. Citado na p. 20.

NIST. **Round 2 Submissions**. Inglês. NIST. Jan. 2017. Disponível em: <https://csrc.nist.gov/Projects/post-quantum-cryptography/round-2-submissions>. Acesso em: 15 jul. 2020. Citado na p. 20.

NIST. **Round 3 Submissions**. Inglês. NIST. Jan. 2017. Disponível em: <https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions>. Acesso em: 15 jul. 2020. Citado nas pp. 20, 54, 69, 87.

NIST. **Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptogrphy Standardization Process**. 2016. Disponível em: <https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf>. Acesso em: 27 jan. 2020. Citado nas pp. 15, 21, 22.

NIST. **Workshops and Timeline**. 2020. Acesso em: 10 fev. 2020. Citado na p. 16.

PEIKERT, Chris. A Decade of Lattice Cryptography. **Foundations and Trends® in Theoretical Computer Science**, v. 10, n. 4, p. 283–424, 2016. ISSN 1551-305X. DOI: 10.1561/0400000074. Disponível em: <http://dx.doi.org/10.1561/0400000074>. Citado nas pp. 43, 45, 46.

PERELMUTER, G. **Futuro Presente: O MUNDO MOVIDO À TECNOLOGIA**. [*S.l.*]: EDITORA NACIONAL, 2019. ISBN 9788504021318. Citado nas pp. 15, 17.

SAMOKHINA, M.; TRUSHINA, O. Code-Based Cryptosystems Evolution. *In:* 2017 IVth International Conference on Engineering and Telecommunication (EnT). [*S.l.*: *s.n.*], nov. 2017. P. 15–17. DOI: 10.1109/ICEnT.2017.10. Citado nas pp. 26, 27, 31.

SENDRIER, N. Code-Based Cryptography: State of the Art and Perspectives. **IEEE Security Privacy**, v. 15, n. 4, p. 44–50, 2017. ISSN 1540-7993. DOI: 10.1109/MSP.2017.3151345. Citado nas pp. 20, 23, 24, 27.

SHOR, P. W. Algorithms for quantum computation: discrete logarithms and factoring. *In:* PROCEEDINGS 35th Annual Symposium on Foundations of Computer Science. [*S.l.*: *s.n.*], 1994. P. 124–134. Citado na p. 15.

SINGH, Harshdeep. Code based Cryptography: Classic McEliece. **arXiv preprint arXiv:1907.12754**, 2019. Citado nas pp. 25–28, 30.

STALLINGS, William. **Criptografia e segurança de redes: princípios e práticas**. Edição: •. [*S.l.*]: Pearson Education do Brasil, 2015. Citado nas pp. 17, 18.

STINSON, Douglas. **Cryptography : theory and practice**. Boca Raton: CRC Press, Taylor & Francis Group, 2019. ISBN 978-1138197015. Citado na p. 28.

SUHAIL, Sabah *et al.* **On the Role of Hash-based Signatures in Quantum-Safe Internet of Things: Current Solutions and Future Directions**. Abr. 2020. Disponível em: <https://www.researchgate.net/publication/340859654_On_the_Role_of_Hash-based_Signatures_in_Quantum-Safe_Internet_of_Things_Current_Solutions_and_Future_Directions>. Citado na p. 36.

TEAM Keccak. [*S.l.*: *s.n.*]. Disponível em: <https://keccak.team/software.html>. Acesso em: 17 set. 2021. Citado na p. 87.

XIN WANG, Bo Yang. An improved signature model of multivariate polynomial public key cryptosystem against key recovery attack. **Mathematical Biosciences and Engineering**, v. 16, 2019. ISSN 1551-0018. DOI: http://dx.doi.org/10.3934/mbe.2019388. Citado na p. 48.

XKCP. **Team Keccak**. [*S.l.*: *s.n.*]. Disponível em: <https://github.com/XKCP/XKCP>. Acesso em: 17 set. 2021. Citado na p. 87.

Apêndices

APÊNDICE A – DADOS DOS GRÁFICOS

Este apêndice, contém todos os dados utilizados para realizar os gráficos.

A.1 ALGORITMOS PÓS-QUÂNTICOS

A.1.1 McEliece

Quadro 14 – Tempo Médio de Geração de Chaves em Segundos do McEliece

Algoritmo	Tempo
mceliece348864	0,1342592
mceliece348864f	0,0552484
mceliece460896	0,3233417
mceliece460896f	0,1338792
mceliece6688128	0,9233591
mceliece6688128f	0,2272384
mceliece6960119	0,7974044
mceliece6960119f	0,2340874
mceliece8192128	0,6580356
mceliece8192128f	0,2559792

A.1.2 SPHINCS

A.1.2.1 Geração de Chaves

Quadro 15 – Tempo Médio de Geração de Chaves SPHINCS Haraka em Segundos

Implementação	Tempo
128f-robust	0,00539853
128f-simple	0,00569781
128s-robust	0,31444310
128s-simple	0,16426114
192f-robust	0,01073959
192f-simple	0,01011196
192s-robust	0,47373549
192s-simple	0,23957483
256f-robust	0,02293617
256f-simple	0,01296078
256s-robust	0,31459284
256s-simple	0,16282864

Quadro 16 – Tempo Médio de Geração de Chaves SPHINCS SHA-256 em Segundos

Implementação	**Tempo**
128f-robust	0,00768846
128f-simple	0,00385763
128s-robust	0,19315596
128s-simple	0,09992493
192f-robust	0,00754211
192f-simple	0,00778802
192s-robust	0,30035314
192s-simple	0,15035632
256f-robust	0,02272734
256f-simple	0,00745567
256s-robust	0,30747349
256s-simple	0,09569765

Quadro 17 – Tempo Médio de Geração de Chaves SPHINCS SHAKE-256 em Segundos

Implementação	**Tempo**
128f-robust	0,00765125
128f-simple	0,00492693
128s-robust	0,32992644
128s-simple	0,17149836
192f-robust	0,00925721
192f-simple	0,00640088
192s-robust	0,48263606
192s-simple	0,24918054
256f-robust	0,02257183
256f-simple	0,02174492
256s-robust	0,31837969
256s-simple	0,16707884

A.1.2.2 Assinatura

Quadro 18 – Tempo Médio de Assinatura SPHINCS Haraka em Segundos

Implementação	**1KB**	**100KB**	**1MB**
128f-robust	0,11982571	0,12413180	0,15553412
128f-simple	0,06396973	0,06726082	0,09848272
128s-robust	2,36857049	2,52924984	2,46977248
128s-simple	1,26249337	1,26698205	1,30534160
192f-robust	0,21522166	0,21478997	0,24705642
192f-simple	0,11275134	0,12066335	0,14877895
192s-robust	4,62712649	4,62040061	4,68573708
192s-simple	2,45870075	2,43108719	2,56766300
256f-robust	0,43921674	0,45075224	0,47956148
256f-simple	0,22943418	0,24024089	0,26687572
256s-robust	4,51984302	4,57848205	4,63466063
256s-simple	2,51992365	2,60362007	2,62720880

Quadro 19 – Tempo Médio de Assinatura SPHINCS SHA-256 em Segundos

Implementação	**1KB**	**100KB**	**1MB**
128f-robust	0,07689085	0,08333894	0,08785778
128f-simple	0,03705761	0,03806224	0,04727435
128s-robust	1,44694636	1,45426267	1,46804538
128s-simple	0,74128078	0,74251045	0,74982939
192f-robust	0,12731754	0,12587823	0,13361692
192f-simple	0,06519111	0,06795911	0,07622660
192s-robust	2,73947569	2,73862648	2,65741266
192s-simple	1,39400931	1,36876055	1,36292085
256f-robust	0,38610303	0,37526932	0,38529021
256f-simple	0,13094104	0,13249048	0,14273361
256s-robust	3,44798886	3,40527359	3,43475480
256s-simple	1,23624657	1,22551264	1,22930562

Quadro 20 – Tempo Médio de Assinatura SPHINCS SHAKE-256 em Segundos

Implementação	**1KB**	**100KB**	**1MB**
128f-robust	0.12657042	0.12709603	0.13601406
128f-simple	0.06629433	0.06595092	0.07476972
128s-robust	2.44323263	2.47060753	2.51403992
128s-simple	1.32831090	1.27992749	1.29759260
192f-robust	0.20726502	0.20628630	0.21597261
192f-simple	0.10833983	0.10937555	0.13188274
192s-robust	4.18112542	4.20877420	4.17156121
192s-simple	2.20259692	2.21586616	2.23879205
256f-robust	0.39722789	0.40720386	0.41118044
256f-simple	0.23457984	0.21442184	0.21980170
256s-robust	3.58563044	3.58840250	3.60457617
256s-simple	1.94659953	1.93552913	1.92977504

A.1.2.3 Verificação

Quadro 21 – Tempo Médio de Verificação SPHINCS Haraka em Segundos

Implementação	**1KB**	**100KB**	**1MB**
128f-robust	0,00753184	0,00916740	0,02526977
128f-simple	0,00413059	0,00565554	0,02125549
128s-robust	0,00288291	0,00501370	0,02046859
128s-simple	0,00160345	0,00339688	0,01866780
192f-robust	0,01143180	0,01378521	0,02904171
192f-simple	0,00584306	0,00765412	0,02366604
192s-robust	0,00452499	0,00598270	0,02211555
192s-simple	0,00245109	0,00401338	0,02014579
256f-robust	0,01207668	0,01452132	0,03039106
256f-simple	0,00621892	0,00805742	0,02376873
256s-robust	0,00663669	0,00831066	0,02475604
256s-simple	0,00357101	0,00530958	0,02169823

Quadro 22 – Tempo Médio de Verificação SPHINCS SHA-256 em Segundos

Implementação	**1KB**	**100KB**	**1MB**
128f-robust	0,00441770	0,00499580	0,00929163
128f-simple	0,00213924	0,00269714	0,00692407
128s-robust	0,00154965	0,00200908	0,00624277
128s-simple	0,00075969	0,00118959	0,00570408
192f-robust	0,00734412	0,00731670	0,01159107
192f-simple	0,00341063	0,00395562	0,00826774
192s-robust	0,00273265	0,00300616	0,00735587
192s-simple	0,00116047	0,00174467	0,00594735
256f-robust	0,01281237	0,01127944	0,01584555
256f-simple	0,00350609	0,00388988	0,00894324
256s-robust	0,00504590	0,00553956	0,01009947
256s-simple	0,00177240	0,00213282	0,00654550

Quadro 23 – Tempo Médio de Verificação SPHINCS SHAKE-256 em Segundos

Implementação	**1KB**	**100KB**	**1MB**
128f-robust	0,00744852	0,00722563	0,01222720
128f-simple	0,00384857	0,00417128	0,00907084
128s-robust	0,00249053	0,00309994	0,00784166
128s-simple	0,00134745	0,00176796	0,00622061
192f-robust	0,01123061	0,01164370	0,01672613
192f-simple	0,00554797	0,00600227	0,01125766
192s-robust	0,00363527	0,00424840	0,00884877
192s-simple	0,00189233	0,00235608	0,00677213
256f-robust	0,01089869	0,01153005	0,01635389
256f-simple	0,00575616	0,00698167	0,01040174
256s-robust	0,00525511	0,00568781	0,01105501
256s-simple	0,00273648	0,00342897	0,00774327

A.1.3 NTRU

Quadro 24 – Tempo Médio de Geração de Chaves NTRU em Segundos

Implementação	**Tempo**
ntruhps2048509	0,0291770
ntruhps2048677	0,0633155
ntruhrss701	0,0643954
mceliece460896f	0,0880904

APÊNDICE B – INSTRUÇÕES PARA COMPILAÇÃO DOS ALGORITMOS

Este apêndice, contém todas as instruções para compilar os algoritmos usados neste trabalho.

B.1 ALGORITMO DE MCELIECE

Para compilar o algoritmo, é necessário baixar o código fonte de (NIST, 2017d). Ao entrar na pasta de alguma versão do algoritmo, temos que dar permissão de execução para o arquivo *build*. Na sequência, executamos o comando *make* no terminal, onde executará o arquivo *build* para compilar o algoritmo. Neste processo, pode ocorrer um erro informando que o arquivo *libkeccak.a.headers/SimpleFIPS202.h* não existe. O algoritmo usa a biblioteca *libkeccac* que não vem instalada no sistema, e portanto, foi preciso instalar. A biblioteca pode ser encontrada em (TEAM. . . , 2021) em XKCP e onde também possui um repositório no GitHub disponível em (XKCP, 2021).

No linux, foi descompactado o *zip* do McEliece em */usr/local/McEliece*. Em seguida, foi preciso baixar o fonte do *XKCP* e compilá-lo e instalar as bibliotecas no sistema utilizando os seguintes comandos:

```
$ make generic64/libXKCP.a
$ cp bin/generic64/* /usr/lib/
$ ln -s /usr/lib/libXKCP.a /usr/local/include/libkeccak.a
$ ln -s /usr/lib/libXKCP.a.headers/ /usr/local/include/libkeccak.a.headers
```

Após instaladas estas dependências, basta compilar o algoritmo utilizando o *makefile* presente nos arquivos e executá-lo utilizando o executável chamado *run*.

B.2 SPHINCS

Através dos arquivos disponibilizados em (NIST, 2017d) referentes a este esquema, foi realizado o download do código. Após obter o algoritmo, é necessário compilá-lo. Isso pode ser feito utilizando o *makefile* presente entre os arquivos. Ao utilizar o comando “make all”, também será criado um executável chamado *benchmark* na pasta *test*. O processo de compilação do SPHINCS não demandou bibliotecas extras.

B.3 NTRU

Através dos arquivos disponibilizados em (NIST, 2017d) referentes a este esquema, foi realizado o download do código fonte. Na sequência, foram compilados os códigos disponíveis na pasta "Optimized_Implementation". Através do arquivo *makefile*, foi possível escolher qual das variante do algoritmo desejou-se compilar. Foram compi-

ladas as variantes: *ntruhps2048509*, *ntruhps2048677*, *ntruhrss701* e *ntruhps4096821*. O processo de compilação do NTRU não demandou bibliotecas extras.

Anexos

Estudo de Algoritmos Criptográficos Pós-Quânticos e Seus Custos

Daniel Boso[1]

[1]Departamento de Informática e Estatística
Universidade Federal do Santa Catarina (UFSC)
Florianópolis – SC – Brasil

daniel.boso@grad.ufsc.br

Abstract. *A study of post-quantum cryptography algorithms and its costs is presented through the monitoring of the standardization process that NIST is carrying out. Hence, it was possible to carry out a study of some algorithms that are in the standardization phase and that already passed the first elemination stages. This study aimed to understand how these algorithms work and its costs. The NIST process is not over yet, but we can already see improvements in some algorithms, where the creators managed to increace efficiency as well as decrease the size of keys and signature.*

Resumo. *Apresenta-se um estudo dos algoritmos criptográficos pós-quânticos e seus custos através do acompanhamento do processo de padronização que o NIST está realizando. Com isso, conseguiu-se realizar um estudo de alguns algoritmos que estão em fase de padronização e que já passaram das primeiras etapas eliminatórias. Este estudo visou entender como estes algoritmos funcionam e seus custos. O processo do NIST ainda terminou, mas já se vê melhorias em alguns algoritmos, onde os criadores conseguiram aumentar a eficiência como também diminuir o tamanho das chaves e assinatura.*

1. Introdução

Quando um computador quântico de larga escala for construído, eles irão comprometer a segurança de muitos algoritmos criptográficos de chave pública comumente usados [NIST 2016]. Alguns especialistas até preveem que dentro dos próximos 20 anos ou mais, um computador suficientemente poderoso computacionalmente será construído para quebrar essencialmente todos os esquemas de chave pública hoje em uso [Chen et al. 2016].

Uma grande comunidade internacional se interessou para abordar a questão da segurança da informação em um computador quântico futuro, na esperança que nossa infraestrutura de chave pública possa continuar funcionando utilizando novas primitivas de criptografia pós-quântica [Chen et al. 2016].

O NIST lançou um vasto programa de padronização de criptografia pós quântica e seu propósito é solicitar, avaliar e padronizar um ou mais algoritmos de criptografia de chave pública [Dragoi et al. 2018]. Chamamos de criptografia pós-quântica as técnicas de criptografia que não sucumbam a ataques realizados por computadores quânticos, que caso venham a se tornar viáveis economicamente, podem ameaçar a privacidade das comunicações online [Perelmuter 2019].

2. Criptografia Pós Quântica

O objetivo da criptografia pós quântica é desenvolver sistemas criptográficos que sejam seguros contra computadores clássicos e quânticos, e possam interoperar com protocolos de comunicação e redes existentes hoje [Chen et al. 2016]. Desta forma, os mesmos aparelhos eletrônicos (ex. Celulares) continuarão funcionando de maneira segura. Bastará apenas trocar o algoritmo criptográfico usado.

A ideia de criptografia pós-quântica foi primeiramente proposta nos anos de 1970 por Stephen Wiesner e Charles H. Bennet da IBM e Gilles Brassard da Universidade de Montreal [Gisin et al. 2002]. Esta, desafia a linha divisória de problemas tratáveis e intratáveis. O exemplo mais significante para isso é algoritmos quânticos eficientes para quebrar sistemas de criptografia na qual são acreditados serem seguros para computadores clássicos [Bernstein et al. 2009].

A busca por algoritmos considerados resistentes a ataques de computadores clássicos e quânticos focou em algoritmos de chave pública pois a construção de um computador quântico de grande escala tornaria muitos destes esquemas de chaves públicas inseguros. Em particular, isso inclui aquelas baseadas na dificuldade de fatoração de inteiros, como RSA, como também aquelas baseadas na dificuldade do problema do logaritmo discreto [Chen et al. 2016].

O que é pretendido na padronização do NIST é deixado claro em [NIST 2017a]. É pretendido que os novos padrões de criptografia de chave pública especifiquem uma ou mais assinaturas digitais adicionais não secretas e disponível publicamente, criptografia de chave pública e algoritmos de estabelecimento de chaves que estão disponíveis em todo o mundo e são capazes de proteger bem informações governamentais confidenciais no futuro próximo, inclusive após o advento dos computadores quânticos.

Os algoritmos submetidos para a primeira rodada do NIST podem ser encontrados em [NIST 2017b]. Também podemos ver em [NIST 2017c] os algoritmos da segunda rodada bem como da terceira rodada em [NIST 2017d]. Nestas referências você pode encontrar arquivos referentes aos algoritmos, seu website, autores e comentários.

Segundo [Sendrier 2017], as principais técnicas de criptografia pós quânticas disponíveis são: criptografia baseada em código (*Code-based Cryptography*), criptografia baseada em hash (*Hash-based Cryptography*), criptografia baseada em reticulados (*Lattice-based Cryptography*) e criptografia polinomial multivariável (*Multivariate Polynomial Cryptography*).

Uma das mais antigas proposições de algoritmos criptográficos resistente a computadores quânticos faz referência a McEliece em 1978 [McEllice 1978], que propôs um sistema de criptografia de chave pública baseada em teoria de código. Tal proposição se beneficia de um algoritmo realmente eficiente como também possui uma base matemática forte [Dragoi et al. 2018]. Desde a contribuição de McEliece, outros sistemas baseados em correção de erros foram propostos. Embora os algoritmos sejam rápidos, a maior parte primitiva de algoritmos baseados em código sofrem por terem tamanho de chaves grandes [Chen et al. 2016].

Assinaturas baseadas em hash são assinaturas digitais construídas usando funções de hash. Sua segurança, até contra ataques quânticos, é bem entendida. Muitos dos mais

eficientes esquemas de assinaturas baseados em hash tem a desvantagem de que o assinante deve manter um registro do número exato das mensagens assinadas anteriormente, e qualquer erro neste registro irá resultar em insegurança. Pode produzir somente um limitado número de assinaturas [Chen et al. 2016].

Construções criptográficas baseadas em reticulados são baseadas na dificuldade de resolver problemas envolvendo reticulados, o mais básico é o problema do caminho mínimo [Bernstein et al. 2009]. Para [Bernstein et al. 2009], criptografia baseada em reticulado mantém uma grande promessa para a criptografia pós-quântica pois eles desfrutam de provas de segurança muito fortes, baseadas na dificuldade de pior caso, implementações relativamente eficientes, bem como grande simplicidade.

Criptografia polinomial multivariada é o estudo de sistemas de criptografia de chave pública onde a função de sentido único toma a forma de um mapa polinomial quadrático sobre um corpo finito [Bernstein et al. 2009].

Acredita-se que estas técnicas resistem a computadores clássicos e quânticos [Bernstein et al. 2009].

3. Testes

Os testes foram realizados em um computador com Ubuntu 20.04.3 LTS, processador AMD® Ryzen 7 5800x 8 núcleos e 16 threads, 32GB de memória e uma placa de vídeo NVDIA GTX 1650 utilizando o driver NV166.

Foram realizados testes para os algoritmo RSA e os algoritmos pós quânticos McEliece, SPHINCS e NTRU da terceira rodada encontrados em [NIST 2017d]. Cada teste, tanto de geração de chave como também de assinatura, foi executado 10 vezes a fim de gerar uma média.

Para a comparação foi utilizado as variantes mais seguras de cada algritmo, disponíveis na Tabela 1, juntamente com suas abreviações que serão utilizadas nos gráficos.

Tabela 1. Variantes dos algoritmos e abreviações

Variante	**Abreviação**
McEliece-6688128	M-6688128
McEliece-6688128f	M-6688128f
McEliece-6960119	M-6960119
McEliece-6960119f	M-6960119f
McEliece-8192128	M-8192128
McEliece-8192128f	M-8192128f
SPHINCS-Haraka-f-r	SPHINCS-Haraka-f-r
SPHINCS-Haraka-f-s	SPHINCS-Haraka-f-s
SPHINCS-Haraka-s-r	SPHINCS-Haraka-s-r
SPHINCS-Haraka-s-s	SPHINCS-Haraka-s-s
SPHINCS-SHA-256f-robust	S-SHA-f-r
SPHINCS-SHA-256f-simple	S-SHA-f-s
SPHINCS-SHA-256s-robust	S-SHA-s-r
SPHINCS-SHA-256s-simple	S-SHA-s-s
SPHINCS-SHAKE-256f-robust	S-SHAKE-f-r
SPHINCS-SHAKE-256f-simple	S-SHAKE-f-s
SPHINCS-SHAKE-256s-robust	S-SHAKE-s-r
SPHINCS-SHAKE-256s-simple	S-SHAKE-s-s
NTRUHPS-2048509	ntruhps2048509
NTRUHPS-2048677	ntruhps2048677
NTRUHRSS-701	ntruhrss701
NTRUHPS-4096821	ntruhps4096821

Em todos os algoritmos testados foram realizados testes de geração de chaves a fim de obter o tempo médio para gerar uma chave. Neste processo, pudemos descobrir o tamanho das chaves públicas e privadas. Assim, a partir do tempo médio de geração de chaves, podemos calcular a quantidade média de chaves geradas a cada segundo.

A partir da quantidade de chaves geradas a cada segundo, juntamente com o tamanho das chaves, podemos trazer a ideia de armazenamento médio. Quando fala-se sobre espaço de armazenamento, considera-se que estas chaves serão armazenadas em algum sistema de armazenamento. Com isso, multiplicando a quantidade média de chaves geradas e o tamanho delas (lembrando de multiplicar o tamanho da chave pública e privada separadamente e somá-los depois), podemos calcular o espaço médio necessário (neste ambiente de testes) a cada segundo pelos algoritmos.

Para o algoritmo SPHINCS foi realizado testes de assinatura utilizando mensagens de tamanho fixo de 100KB. Para estes mesmos algoritmos também foram realizados testes de verificação.

Para fazer os testes, foram utilizados os algoritmos disponíveis para download em [NIST 2017d].

3.1. Geração de Chaves

Ao observar na Figura 1, podemos notar que o RSA possui uma perda de desempenho expressiva após o RSA-4096. No caso do McEliece, podemos observar que as versões "f" do algoritmo são mais rápidas. Já para o caso do SPHINCS, podemos notar que as versões "f" são mais rápidas que suas respectivas versões "s". Podemos observar também que o algoritmo entre as técnicas pós quânticas que teve melhor desempenho na geração de chaves foi o SPHINCS com a variante SPHINCS-SHA-256-256f-simple (S-SHA-f-s). Já para o caso do RSA, para as versões acima de 7680 bits o RSA apresenta uma piora no desempenho quando comparado com as variantes "f" do SPHINCS.

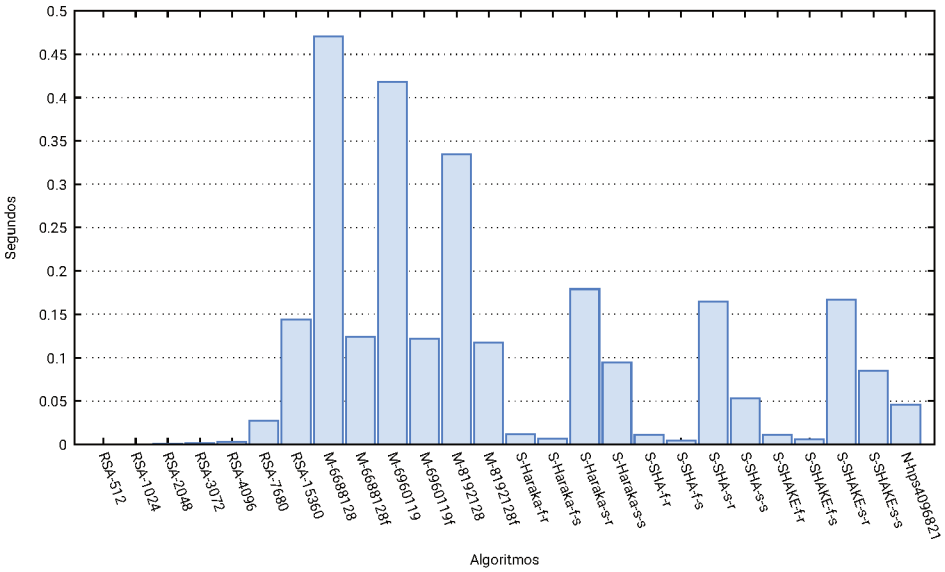


Figura 1. Geração de Chaves

Na Tabela 2, é apresentado o tamanho das chaves e assinatura dos algoritmos testados. Nota-se que o mceliece8192128 e mceliece8192128f possem a maior chave pública atingindo aproximadamente 1.3MB. Os algoritmos com menores tamanhos de chaves foi o SPHINCS, seguido do NTRU.

Tabela 2. Tamanhos de Chave Pública, Chave Privada e Assinatura em Bytes

Implementação	Chave Pública	Chave Privada	Assinatura
mceliece6688128	1.044.992	13.932	-
mceliece6688128f	1.044.992	13.932	-
mceliece6960119	1.047.319	13.948	-
mceliece6960119f	1.047.319	13.948	-
mceliece8192128	1.357.824	14.120	-
mceliece8192128f	1.357.824	14.120	-
sphincs-256s	64	128	29.792
sphincs-256f	64	128	49.856
ntruhps2048509	699	935	-
ntruhps2048677	930	1234	-
ntruhrss701	1138	1450	-
ntruhps4096821	1230	1590	-

3.2. Assinatura

Para a assinatura, podemos comparar o RSA com o SPHINCS. Entre os algoritmos pós quânticos apresentados, a variante do SPHINCS que obteve o maior desempenho foi a SPHINCS-SHA-256-128f-simple (S-SHA-f-s). Podemos observar que o RSA-15360 obteve um desempenho inferior a algumas variantes do SPHINCS

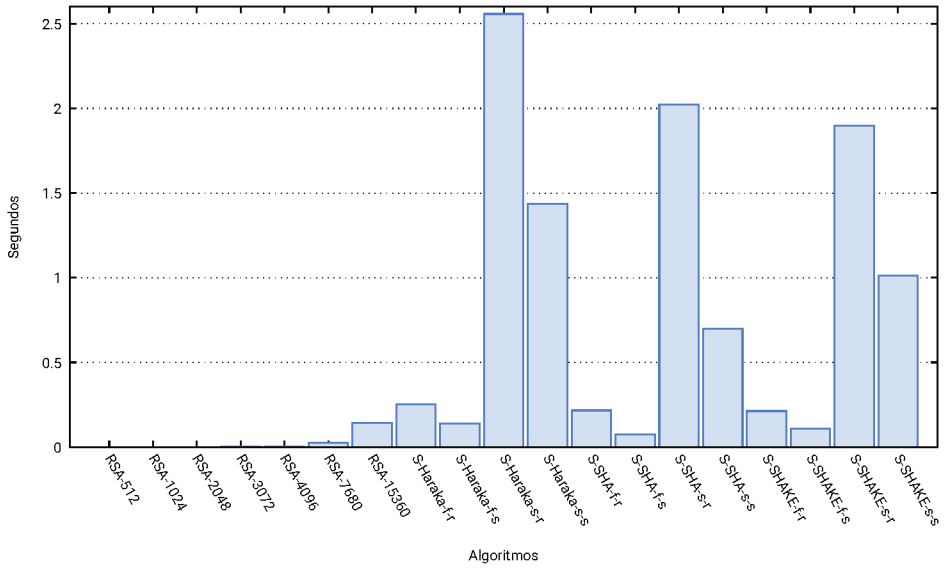


Figura 2. Assinatura

3.3. Verificação

No caso da verificação, a variante do SPHINCS que obteve maior desempenho foi a SPHINCS-SHA-256-128s-simple (S-SHA-s-s). Em comparação com essa variante do SPHINCS, o RSA obteve maior desempenho.

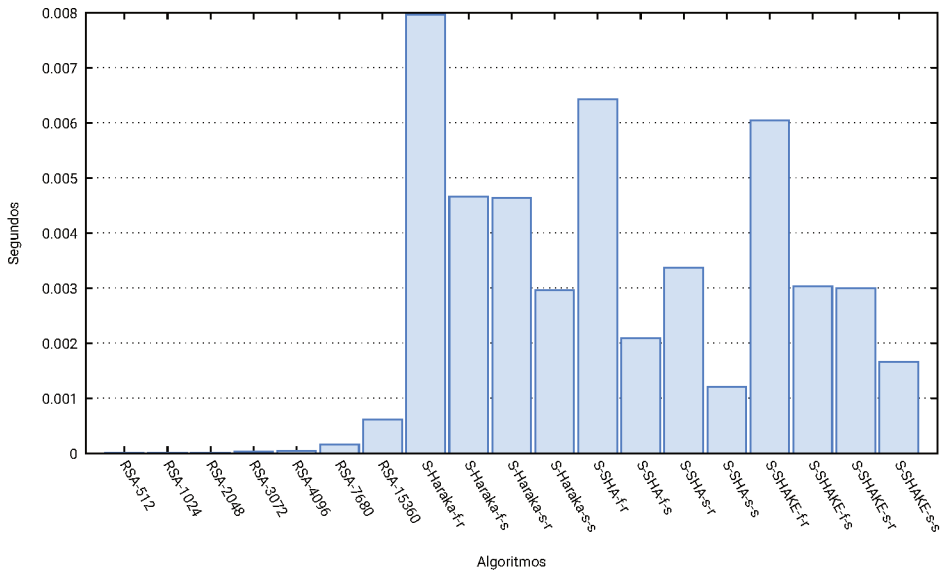


Figura 3. Verificação

4. Conclusões

Este artigo procurou mostrar os testes de desempenho entre as principais e mais relevantes técnicas de criptografia pós-quânticas, mostrando os pontos fortes e fracos de cada uma.

Neste processo, se percebeu a alta complexidade na escolha de um algoritmo "ótimo", visto que os algoritmos apresentados possuem várias variantes onde podem, por sua complexidade, fazer trocas entre tempo e espaço de armazenamento, afetando o comprimento de chaves públicas e privadas bem como tamanho de assinatura.

Percebeu-se também que utilizar algum algoritmo pós-quântico criptográfico, pode ter menor desempenho quando comparado com o algoritmo RSA. Porém, como visto no comparativo das técnicas com o RSA, há variantes que superam o desempenho do RSA em algum aspecto, seja na geração de chaves ou assinatura.

O processo de padronização do NIST tem sido muito importante para coletar algoritmos que são resistentes a computadores quânticos. Neste processo, se observa que houve melhorias nos algoritmos e espera-se que isso aconteça também na medida em que estes algoritmos passem para as fases posteriores.

Sendo assim, antes mesmo de um computador quântico de larga escala ser construído, sabemos que temos outras opções de algoritmos e que ainda estão sendo estudados, talvez com menor desempenho quando comparado a sistemas de criptografia utilizados hoje, porém, mais seguros e resistentes a ataques de computadores quânticos.

Referências

Bernstein, D. J., Buchmann, J., and Dahmen, E., editors (2009). *Post-Quantum Cryptography*. Springer Berlin Heidelberg.

Beullens, W. (2022). Breaking rainbow takes a weekend on a laptop. https://eprint.iacr.org/2022/214.pdf.

Chen, L., Jordan, S., Liu, Y.-K., Moody, D., Peralta, R., Perlner, R., and Smith-Tone, D. (2016). *Report on Post-Quantum Cryptography*. National Institute of Standards and Technology(NIST).

Dragoi, V., Richmond, T., Bucerzan, D., and Legay, A. (2018). Survey on cryptanalysis of code-based cryptography: from theoretical to physical attacks. In *2018 7th International Conference on Computers Communications and Control (ICCCC)*.

Gisin, N., Ribordy, G., Tittel, W., and Zbinden, H. (2002). Quantum cryptography. *Rev. Mod. Phys.*, 74:145–195.

McEllice (1978). A public-key system based on algebraic coding theory. Technical report, Jet Propulsion Lab.

NIST (2016). Submission requirements and evaluation criteria for the post-quantum cryptogrphy standardization process. https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf. Acessado em 27/01/2020.

NIST (2017a). Post-quantum cryptography standardization. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization. Acessado em 15/07/2020.

NIST (2017b). Round 1 submissions. https://csrc.nist.gov/Projects/post-quantum-cryptography/Round-1-Submissions. Acessado em 15/07/2020.

NIST (2017c). Round 2 submissions. https://csrc.nist.gov/Projects/post-quantum-cryptography/round-2-submissions.

NIST (2017d). Round 3 submissions. https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions. Acessado em 15/07/2020.

Perelmuter, G. (2019). *Futuro Presente: O MUNDO MOVIDO À TECNOLOGIA*. EDITORA NACIONAL.

Sendrier, N. (2017). Code-based cryptography: State of the art and perspectives. *IEEE Security Privacy*, 15(4):44–50.