UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS CRATEÚS

CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

MATEUS SOARES BEZERRA

XMSS VS. SPHINCS$^+$: COMPARAÇÃO ENTRE DOIS ESQUEMAS DE ASSINATURAS DIGITAIS PÓS-QUÂNTICOS

CRATEÚS

2020

MATEUS SOARES BEZERRA

XMSS VS. SPHINCS$^+$: COMPARAÇÃO ENTRE DOIS ESQUEMAS DE ASSINATURAS DIGITAIS PÓS-QUÂNTICOS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação do Campus Crateús da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Sistemas de Informação.

Orientador: Prof. Me. Roberto Cabral Rabêlo Filho

CRATEÚS

2020

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

B469x Bezerra, Mateus Soares.
XMSS vs. SPHINCS+: Comparação entre dois Esquemas de Assinaturas Digitais pós-quânticos / Mateus Soares Bezerra. – 2020.
52 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, , Crateús, 2020.
Orientação: Prof. Me. Roberto Cabral Rabêlo Filho.

1. Assinatura digital. 2. Função de resumo. 3. Criptografia pós-quântica. I. Título.

CDD

MATEUS SOARES BEZERRA

XMSS VS. SPHINCS^{+}: COMPARAÇÃO ENTRE DOIS ESQUEMAS DE ASSINATURAS DIGITAIS PÓS-QUÂNTICOS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação do Campus Crateús da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Sistemas de Informação.

Aprovada em: 20 de Agosto de 2020

BANCA EXAMINADORA

Prof. Me. Roberto Cabral Rabêlo Filho (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Sidartha Azevedo Lobo de Carvalho
Universidade Federal do Ceará (UFC)

Profª Drª Ana Karina Dourado Salina de Oliveira
Universidade Federal de Mato Grosso do Sul (UFMS)

À minha família, por sua capacidade de acreditar em mim e investir em mim. Mãe, seu cuidado e dedicação foi que deram, em alguns momentos, a esperança para seguir. Pai, sua presença significou segurança e certeza de que não estou sozinho nessa caminhada.

AGRADECIMENTOS

Ao professor Me. Roberto Cabral Rabêlo Filho, pela ótima orientação.

A professora Me. Lisieux Marie Marinho Dos Santos Andrade no auxílio da escrita ministrada na cadeira de PPCT.

Aos professores participantes da banca examinadora pelo tempo, pelas valiosas colaborações e sugestões.

Aos colegas da turma de graduaçãao, pelas reflexões, críticas e sugestões.

A minha família.

"Nothing is true, everything is permitted."

(The Assassin's Creed)

RESUMO

Com o avanço da tecnologia cada dia mais inserido no nosso dia a dia, surgiu a necessidade de utilizar técnicas mais sustentáveis, como a substituição de documentos físicos em papel por documentos eletrônicos. Entretanto, é necessário que a segurança dos dados em tais documentos esteja assegurada. Uma das formas de se obter tal segurança é através da assinatura digital. Um esquema de assinatura digital é uma tecnologia indispensável para fornecer certas características essenciais a dados eletrônicos, sendo elas: autenticidade, integridade e irretratabilidade. Diversas aplicações de várias áreas utilizam assinaturas digitais para a segurança de seus dados, tais como: autenticação de usuários, comércio eletrônico, distribuição de chaves de softwares, dentre outras. Porém, com o surgimento dos computadores quânticos, grande parte dos esquemas de assinaturas digitais mais utilizados nos dias de hoje deixariam de ser seguros, pois sua segurança depende da dificuldade de fatorar números inteiros grandes e calcular logaritmos discretos. Para isso, é necessário utilizar esquemas de assinaturas que sejam resistentes à computadores quânticos; os esquemas de assinatura digital baseados em funções de resumo são considerados resistentes aos computadores quânticos. Esse trabalho faz um estudo dos esquemas de assinatura digital baseados em funções de resumo, um já padronizado, XMSS, e apresenta um comparativo com um novo algoritmo candidato a padronização, SPHINCS$^+$.

Palavras-chave: Assinatura digital. Função de resumo. Criptografia pós-quântica.

ABSTRACT

With the advancement of technology, each day more inserted in our daily life, the need arose to use more sustainable techniques, such as the substitution of physical paper documents for electronic documents. However, it is necessary that the security of the data in such documents is ensured. One of the ways to obtain such security is through digital signature. A digital signature scheme is an indispensable technology for providing specific essential characteristics to electronic data, such as: authenticity, integrity, and non-retractability. Many applications in various areas use digital signatures to secure their data, such as user authentication, eCommerce, software key distribution, and more. However, with the emergence of quantum computers, most of today's most widely used digital signature schemes would no longer be secure because their security depends on the difficulty of factoring large integers and calculating discrete logarithms. This requires the usage of signature schemes that are resistant to quantum computers; Hash-based digital signature schemes are considered resistant to quantum computers. This paper makes a study of digital signature schemes based on hash functions, one already standardized, XMSS, and presents a comparison with a new algorithm candidate for standardization, SPHINCS$^+$.

Keywords: Digital signature. Hash functions. Post-quantum cryptography.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

FORS	*Forest of Random Subsets*
NIST	*National Institute of Standards and Technology* / Instituto Nacional de Padrões e Tecnologia
XMSS	*Extended Merkle Signature Scheme* / Esquema de Assinatura de Merkle Estendido
XOR	*Exclusive OR* / OU Exclusivo
AES	*Advanced Encryption Standard* / Padrão de Encriptação Avançado
DSA	*Digital Signature Algorithm* / Algoritmo de Assinatura Digital
ECDSA	*Elliptic Curve Digital Signature Algorithm* / Algoritmo de Assinatura Digital de Curvas Elípticas
ANSI	*American National Standards Institute* / Instituto Nacional Americano de Padrões
OTS	*One-Time Signature*
MSS	*Merkle Signature Scheme* / Esquema de Assinatura de Merkle
LMS	*Leighton-Micali Signature* / Assinatura de Leighton e Micali

SUMÁRIO

1 INTRODUÇÃO

Com o passar do tempo, a criptografia, que costumava tratar da criação de códigos para mandar mensagens confidenciais, passou a tratar de problemas de autenticação, assinaturas digitais, comércio eletrônico, eleições eletrônicas, moeda digital, entre outros. Podemos afirmar que a criptografia dos dias atuais tenta tratar de problemas de segurança que podem aparecer em qualquer sistema computacional. Katz (2014) diz que a criptografia é uma ciência que busca técnicas de segurança para informações no âmbito digital, de transações e de computação distribuída. Técnicas de criptografia são empregadas para realizar o controle de acesso em sistemas operacionais multiusuário, proteção de software e outros mecanismos para evitar cópias indevidas.

A Segurança da Informação é um ramo da computação que vem conseguindo bastante destaque na academia e na indústria. Cada vez mais se discute a importância de controle de acesso e sigilos, componentes fundamentais que trazem para quem faz uso do meio tecnológico um cenário cheio de particularidades. Esse ambiente tecnológico, não se propõe apenas à facilidades, mas também à circunstâncias para assegurar confiança nas atividades executadas, ainda mais durante nosso período atual, o qual estamos passando pela pandemia do Corona vírus, necessitamos realmente desse ambiente seguro para executar nossas tarefas sendo trabalho remoto, ou estudos remotos. Diversos são os mecanismos desenvolvidos para tentar alcançar tal propósito. A criptografia é um dos elementos primordiais dentro dessas ferramentas, por causa de sua ampla difusão e conjunto de funcionalidades. Uma definição mais contemporânea estende os serviços de segurança oferecidos pela criptografia para além da confidencialidade, incluindo autenticação, verificação de integridade, irretratabilidade entre outros.

Por muito tempo, os algoritmos criptográficos dependiam de um tipo de chave, para manter o segredo da mensagem. Na criptografia de chave privada (ou criptografia simétrica), dois indivíduos que desejassem trocar mensagens, precisam partilhar uma chave privada. Um dos problemas desse tipo de criptografia é que as chaves privadas não podem ser compartilhadas através de um canal inseguro, porque um oponente pode interceptá-las durante o compartilhamento. O envio das chaves deve ser feito por meio de um canal considerado seguro, usando algum serviço confiável para troca de mensagens ou entregue pessoalmente.

A criptografia de chave pública (ou criptografia assimétrica), faz uso de um par de chaves, sendo uma chave pública, que é conhecida abertamente, e a outra uma chave privada, que é conhecida apenas pelo possuidor do par de chaves. Um dos objetivos da criptografia

assimétrica é possibilitar que dois indivíduos troquem mensagens secretas através de um canal inseguro. E, além de oferecer sigilo, a criptografia assimétrica também consegue fornecer a assinatura digital das mensagens.

Esquemas de assinatura digital são uma das técnicas criptográficas mais utilizadas por aplicações, que necessitam de autenticidade, integridade e irretratabilidade. Atualmente, uma parte dos esquemas de assinaturas digitais utilizados têm sua segurança baseada na dificuldade de fatorar grandes números inteiros ou calcular logaritmos discretos. Shor (1994) propôs um algoritmo quântico para fatoração de números inteiros grandes e calculo de logaritmos discretos em corpos finitos em tempo hábil, utilizando operações simples de um computador quântico. O aparecimento de computadores quânticos potentes ameaçará a segurança dos esquemas de criptografia mais utilizados atualmente, pois se o algoritmo de Shor, que pode fatorar números inteiros e calcular logaritmos discretos em tempo hábil, for implementado, atacantes equipados com um computador quântico terão uma grande possibilidade de quebrar a segurança de alguns dos esquemas de assinatura digital e outras ferramentas da criptografia utilizadas atualmente.

A criptografia pós-quântica busca construir técnicas criptográficas que permanecerão seguras mesmo no aparecimento de computadores quânticos. Pesquisas referentes à criptografia pós-quântica têm ganhado destaque, principalmente por causa da competição para algoritmos pós-quânticos proposta pelo *National Institute of Standards and Technology* / Instituto Nacional de Padrões e Tecnologia (*NIST*) (2016), que busca padronizar algoritmos pós-quânticos de assinatura digital e por isso diversos esquemas voltaram a ser analisados. Assim, fazendo surgir esquemas como: Esquemas de assinatura digital baseados em funções de resumo, criptografia baseada em sistemas multivariados e criptografia baseada em reticulados e baseado em provas de conhecimento Zero não interativas.

O aparecimento de um computador quântico afeta a maioria dos esquemas de assinaturas digitais usados atualmente e portanto, é de extrema importância fornecer algoritmos pós-quânticos práticos. Visto que os esquemas baseados em funções de resumo são considerados fortes candidatos para resolver essa problemática, este trabalho busca fazer um estudo das principais características dos esquemas baseados em funções de resumo, em especial o Esquema de Merkle Estendido que já é alvo de padronização e o SPHINCS$^+$, um esquema de assinatura baseada em função de resumo proposto recentemente, que conseguiu avançar na segunda fase de padronização da competição organizada pelo *NIST* mostrando-se um candidato promissor a esquema de assinatura digital pós-quântico seguro.

1.1 Objetivos

Este trabalho busca realizar um estudo e comparação de alguns esquemas de assinatura digital baseados em funções de resumos, o *Extended Merkle Signature Scheme* / Esquema de Assinatura de Merkle Estendido (*XMSS*) e o esquema SPHINCS$^+$. Sendo o *XMSS* um esquema já padronizado e o SPHINCS$^+$ um novo esquema candidato a padronização.

1.1.1 Objetivos Específicos

- Estudar os algoritmos *XMSS* e SPHINCS$^+$;
- Fazer uma explicação simplificada sobre as partes mais importantes dos esquemas;
- Descrever as principais diferenças entre os esquemas;
- Fazer uma comparação entre os algoritmos em: desempenho, tamanho de chaves, tamanho da mensagem e tamanho da assinatura;

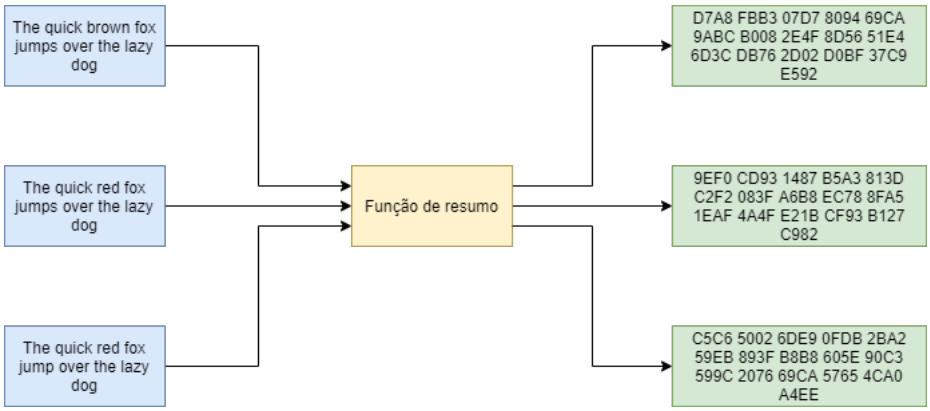
2 REVISÃO BIBLIOGRÁFICA

Neste capítulo é feita uma revisão da literatura científica acerca de assinaturas digitais pós-quânticas, onde estão apresentadas as principais características e recursos necessários para melhor entendimento deste trabalho.

2.1 Funções de resumo criptográfica

Função hash ou função de resumo criptográfico é um algoritmo que mapeia um certo dado de tamanho qualquer, por exemplo, um arquivo ou uma mensagem, em um valor de resumo de tamanho fixo (PAAR; PELZL, 2009). As funções de resumo foram apresentadas como mais um dos inúmeros mecanismos da criptografia e passaram a ser utilizadas como ferramentas em aplicações de segurança para realizar, por exemplo, armazenamento seguro de senhas, assinaturas digitais, etc. Essas funções podem ser usadas para resolver problemas que envolvem autenticação e integridade de dados. Dang (2008) diz que uma função de resumo deve ser fácil de ser implementada e fácil de ser utilizada, deve ser eficiente, deve ser resistente à ataques cripto-analíticos já conhecidos e deve possuir certas propriedades que serão mostradas a seguir. Uma função de resumo mapeia uma mensagem, esta mensagem muitas vezes é representada por uma cadeia binária de tamanho finito e arbitrário, em um valor de resumo que é uma cadeia binária de tamanho fixo. A Figura 1 ilustra o funcionamento da função de resumo SHA-2.

Figura 1 – Ilustração do funcionamento de uma função de resumo (SHA-2)



Fonte – Próprio autor

2.1.1 Propriedades e segurança das funções de resumo

Ferguson, Schneier e Kohno (2011) apresentaram certas propriedades das funções e destacaram resistência à pré-imagem, resistência à segunda pré-imagem e resistência à colisão como sendo algumas das propriedades mais importantes das funções de resumo.

- Resistência à pré-imagem: dado um valor de resumo r e a função de resumo h que gerou esse valor, deve ser computacionalmente difícil encontrar alguma mensagem M tal que $h(M) = r$.
- Resistência à segunda pré-imagem: dada uma mensagem qualquer M e uma função de resumo h, deve ser computacionalmente difícil encontrar uma outra mensagem M', tal que $M \neq M'$ e $h(M) = h(M')$.
- Resistência à colisão: dado uma função de resumo h, deve ser computacionalmente difícil encontrar duas mensagens quaisquer M e M', tal que $M \neq M'$ e $h(M) = h(M')$.

A propriedade de resistência a colisões é o motivo de funções de resumo serem utilizadas por esquemas de assinatura. Toda e qualquer função de resumo tem colisões, pois o tamanho do valor de resumo é fixo, sendo assim finito, enquanto o número de mensagens que pode ser usado é infinito, então, em algum momento, duas mensagens diferentes vão ter o mesmo valor de resumo. Por essa limitação no tamanho do valor de resumo, não existe uma função de resumo sem colisões. É necessário apenas que seja computacionalmente difícil encontrar tais colisões para um dada função de resumo.

A segurança de uma função de resumo se baseia exatamente nas propriedades citadas acima. Uma forma de tentar quebrar a segurança das funções de resumo é por meio de ataques às suas propriedades. Um ataque bem conhecido é chamado Ataque do Aniversário que têm como ênfase a propriedade a resistência à colisão, que baseia-se em encontrar duas mensagens diferentes que produzam o mesmo valor de resumo. Ele recebe esse nome do Paradoxo do Aniversário, que consiste em determinar a probabilidade de que dado um número de k pessoas em uma sala, existam duas pessoas que fazem aniversário no mesmo dia. Com esse ataque há uma alta probabilidade de encontrar uma colisão em $2^{l/2}$ passos, onde l é o tamanho(em *bits*) do valor de resumo.

Veremos agora três das funções de resumo utilizadas atualmente e que são também utilizadas pelos esquemas de assinatura digital escolhidos neste trabalho.

2.1.2 SHA-2

O SHA-2 é um conjunto de funções de resumo. O SHA-2 é considerado uma "família"de algoritmos, pois a diferença mais notável entre eles é a segurança provida por cada algoritmo. Outra diferença a se destacar é o tamanho de bloco e de palavra. Os algoritmos SHA-224 e SHA-256 trabalham com blocos de 512 *bits* e palavras de 32 *bits* enquanto os algoritmos SHA-384 e SHA-512 trabalham com blocos de 1024 *bits* e palavras de 64 *bits*.

O funcionamento dos algoritmos do SHA-2 pode ser dividido em duas partes: o pré-processamento e computação do valor de resumo. A parte de pré-processamento consiste em preencher uma mensagem, dividir essa mensagem preenchida em blocos de tamanho fixo (32 ou 64 *bits* dependendo de qual algoritmo é utilizado) e, por fim, estabelecer certos valores que serão utilizados para computar o valor de resumo.

Cada uma das funções de resumo trabalham com seis funções lógicas que operam sobre as palavras de 32 *bits* para SHA-224 e SHA-256, ou 64 *bits* para SHA-384 e SHA-512. Essas funções lógicas transformam as palavras, utilizando principalmente o operador *Exclusive OR* / OU Exclusivo (*XOR*), em novas palavras de mesmo tamanho.

A parte de preenchimento serve para garantir que a mensagem preenchida seja múltipla de 512 ou 1024 *bits* dependendo do algoritmo. O preenchimento pode ser feito antes do valor de resumo começar a ser computado ou durante a computação. Mais detalhes a respeito da função de resumo SHA-2 podem ser encontrados no documento que o define, PUB (2012).

2.1.3 SHA-3

O SHA-3, assim como o SHA-2, é um conjunto de funções de resumo que são similares entre si. Todas as funções da família do SHA-3 derivam do algoritmo *KECCAK* proposto por BERTONI *et al.*, que foi vencedor da competição proposta pelo *NIST* para a seleção do algoritmo para o SHA-3. A família do SHA-3 é composta de seis algoritmos, sendo quatro deles funções de resumo e os outros dois funções de saída variável (*extendable-output*). Todas as seis funções utilizam uma estrutura chamada construção esponja, proposta por Bertoni et al. (2007).

Uma função de saída variável é uma função que age sobre cadeias de *bits* na qual o tamanho para a saída de tais funções pode ser estendida para qualquer tamanho. As duas funções variáveis do SHA-3 foram as primeiras funções variáveis a serem padronizadas pelo *NIST*.

Cada uma das funções usa uma permutação sobre a construção esponja. Essa permutação é chamada Keccak-p e opera com dois parâmetros: o comprimento fixo das cadeias que são permutadas e o número de rodadas, que é o número de iterações internas de uma transformação. Uma rodada do Keccak-p é uma sequência de cinco transformações, chamadas etapas de mapeamento.

As cinco etapas de mapeamento que compõem uma rodada do Keccak-p são denotadas por θ, ρ, π, χ e ι. O algoritmo de cada etapa recebe como entrada um array de estado e retorna como saída esse array de estado atualizado. A etapa ι tem uma entrada adicional, um inteiro que indica o índice da rodada.

- **Etapa de mapeamento** θ. O efeito da etapa θ é aplicar uma operação *XOR* entre cada palavra do estado com a paridade de duas colunas do estado.
- **Etapa de mapeamento** ρ. Nesta etapa cada palavra do estado é rotacionada uma quantidade fixa de *bits*.
- **Etapa de mapeamento** π. O efeito da etapa de mapeamento π é embaralhar as palavras do estado. Ela promove uma difusão a longo prazo dentro das rodadas, a fim de evitar que padrões sejam explorados em determinados ataques.
- **Etapa de mapeamento** χ. O efeito da etapa de mapeamento χ é aplicar uma operação *XOR* em cada palavra do estado com a saída de uma função não linear aplicada a duas palavras da mesma linha do estado.
- **Etapa de mapeamento** ι. A etapa de mapeamento ι consiste na aplicação de uma operação *XOR* entre o elemento de um estado com um valor constante. O efeito desta etapa consiste em modificar alguns *bits* da primeira palavra do estado, de acordo com o índice da rodada. As outras 24 palavras do estado nunca são afetadas diretamente pela etapa de mapeamento ι.

A construção esponja é uma estrutura para criar funções que operam sobre dados binários e que tem saídas de tamanho arbitrário. A construção usa três componentes, uma função de permutação, um parâmetro de taxa de *bits* e uma regra de preenchimento, juntos esses componentes dessa construção definem uma função, chamada função esponja.

No SHA-3, as palavras são de 64 bits e a quantidade de rodadas são fixas em 24 para todas as instâncias da família SHA-3. Mais detalhes a respeito da função de resumo SHA-3 podem ser encontrados no (PUB, 2014) que o define.

2.1.4 Haraka

Geralmente as funções de resumo são projetadas tendo como principal requisito a resistência a colisão. Porém, ultimamente tem surgido diversas aplicações que utilizam funções de resumo, mas não requerem que tais funções sejam resistentes a colisão. Um exemplo de aplicação seriam os esquemas de assinatura digital baseado em funções de resumo. Tais esquemas geralmente realizam muitas chamadas às funções de resumo que utilizam, mas essas chamadas só passam entradas de tamanho pequeno para a função de resumo processar. Como a grande maioria das funções de resumo são projetadas para atender resistência a colisão e trabalhar com mensagens de tamanho grande, seu desempenho com entradas pequenas é ruim.

Assim, pensou-se na construção de uma função de resumo especificamente para esse tipo de aplicação. A função de resumo Haraka foi criada com esse propósito. Kölbl et al.(2016) propuseram o Haraka como uma função de resumo que atende apenas as propriedades de resistência a pré-imagem e resistência à segunda pré-imagem, deixando de lado a resistência a colisão e melhorando a performance para entradas fixas de tamanho pequeno justamente para melhorar o desempenho em esquemas de assinatura digital baseados em função de resumo. O Haraka foi projetado baseado nas instruções e rodadas de permutação do algoritmo criptográfico simétrico *Advanced Encryption Standard* / Padrão de Encriptação Avançado (*AES*).

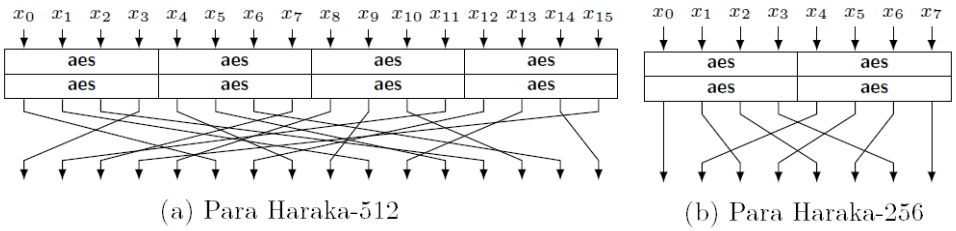
O Haraka foi proposto com duas variantes: Haraka-512 e Haraka-256, ambos com segurança de 256 *bits*. A Figura 2 ilustra componentes principais do Haraka, tanto do 512 quanto do de 256, são permutações que fazem a aplicação da operação *XOR* na entrada que recebem. Essas permutações são feitas utilizando uma função de rodada, sendo aplicada diversas vezes para completar a permutação. Os projetistas decidiram utilizar a função de rodada do *AES* pelo motivo de várias intruções do *AES* já estarem implementadas como instruções nativas de CPUs mais recentes. Para mais detalhes a respeito da função de resumo Haraka podem ser encontrados no documento que o define, Kölbl *et al.* (2016).

2.2 Assinaturas Digitais Clássicas

Paar e Pelzl (2009) sugerem que as assinaturas digitais são uma ferramenta importante para a criptografia de chave pública, amplamente usada atualmente.

Assinaturas digitais tem características semelhantes à assinaturas feitas à mão, onde uma dessas características é a de identificar se uma mensagem é autêntica para um dado usuário,

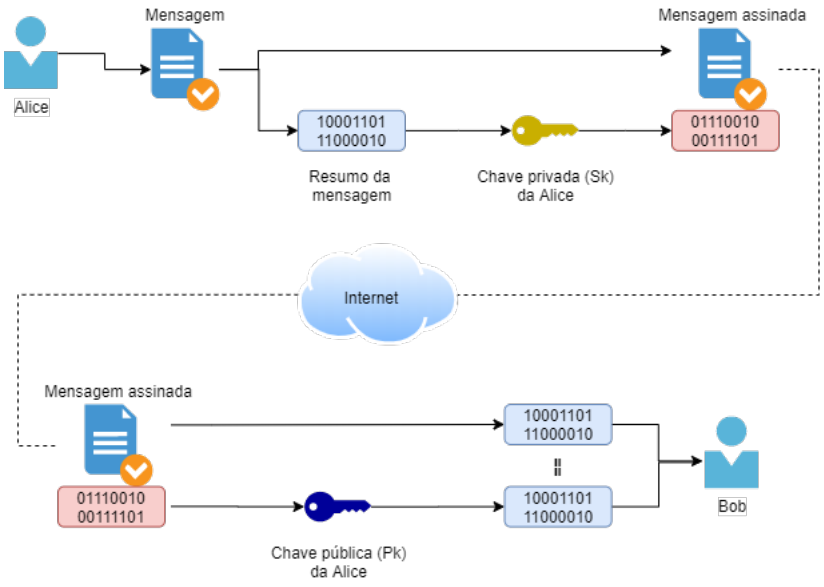
Figura 2 – Função de rodada das permutações para as duas variante do Haraka



Fonte – Adaptado de Kölbl *et al.* (2016)

ou seja, se a mensagem vem realmente de quem alega ter gerado a mensagem. Essa característica é muito importante tanto no mundo real, através de assinaturas à mão, quanto no mundo digital, através de assinatura digital. Assim, como nas assinaturas feitas à mão, é possível garantir a autenticidade de uma assinatura digital. A Figura 3 ilustra o processo de assinatura digital utilizando ferramentas criptográficas, utilizando a criptografia de chave pública, onde a pessoa que assina a mensagem usa uma chave privada e quem vai verificar a mensagem utiliza uma chave pública correspondente.

Figura 3 – Ilustração do funcionamento de uma assinatura digital



Fonte – Próprio autor

As três famílias de algoritmos de chave pública mais populares, que nos permitem construir assinaturas digitais, se baseiam na fatoração de números inteiros, logaritmos discretos e curvas elípticas. Sendo eles o RSA (*Rivest, Shamir, Adleman*) (1978), *Digital Signature*

Algorithm / Algoritmo de Assinatura Digital (*DSA*) (2013) e *Elliptic Curve Digital Signature Algorithm* / Algoritmo de Assinatura Digital de Curvas Elípticas (*ECDSA*) (1999).

Um esquema de assinatura qualquer é composto por três partes principais, a geração de chaves, a geração da assinatura e a verificação da assinatura. De forma simplificada, cada uma dessas partes tem um algoritmo que funcionam da seguinte forma:

- Geração de chaves: o algoritmo produz um par de chaves, uma pública e uma privada, a partir de um parâmetro de segurança k que recebe como entrada;
- Geração da assinatura: o algoritmo produz uma assinatura *sig* a partir de uma mensagem M e de uma chave privada *Sk*, gerada no algoritmo de geração de chaves;
- Verificação da assinatura: o algoritmo recebe como entrada uma mensagem M, a assinatura *sig* da mensagem M e a chave pública *Pk*, geradas nos algoritmos anteriores, e produz um valor booleano como resultado da verificação. Se o valor for *true*, então isso significa que a assinatura é válida e caso contrário, significa que a assinatura é inválida.

2.2.1 RSA

Proposto por Rivest, Shamir e Adleman (1978), o algoritmo criptográfico RSA é a base para o esquema de assinatura digital RSA. Sua segurança advém da dificuldade de fatorar um produto de dois grandes números primos (o problema da fatoração de inteiros). O criptossistema RSA é o esquema de criptografia assimétrica mais utilizado, mesmo com os esquemas de curvas elípticas e logaritmo discreto ganhando espaço.

A geração das chaves no RSA é feita de maneira relativamente simples. Primeiro, deve-se escolher dois números primos grandes p e q. Depois é calculada a multiplicação entre os dois números $n = p \times q$. Após isso, é aplicada a função totiente de Euller (chamada função *phi* ϕ) ao resultado da multiplicação $\phi(n)$. Em seguida, é necessário encontrar outro número aleatório e que deve ser maior que 1 e menor do que o resultado da função $\phi(n)$ e também deve ser primo entre esse resultado da função $\phi(n)$. A chave pública *Pk* é composta pelo resultado da multiplicação entre n e o número aleatório e. A chave privada *Sk* pode ser encontrada de forma que $d \times e \equiv \text{mod } \phi(n)$.

No esquema de chave pública para encriptação, os papeis das chaves são invertidos em comparação com a assinatura digital do RSA. No esquema de encriptação, utiliza-se a chave pública *Pk* do destino para cifrar a mensagem, enquanto que no esquema de assinatura digital, utiliza-se a chave privada *Sk* da origem para assinar a mensagem.

A assinatura da mensagem M (dividida em blocos de tamanho m) é feita da seguinte forma: $sig(m) \equiv m^d \mod n$. Para verificar a assinatura, faz-se da seguinte forma: $temp \equiv sig(m)^e \mod n$. Se $temp \equiv m \mod n$, a assinatura é válida, se $temp \not\equiv m \mod n$, a assinatura é inválida.

Note que foi mostrado apenas um esquema de assinatura digital, a mensagem em si não é encriptada e, portanto, não há confidencialidade. Se tal tarefa for necessária, a mensagem junto com a assinatura devem ser encriptadas, usando algum algoritmo de criptografia simétrica, por exemplo.

2.2.2 Digital Signature Algorithm

O *DSA* é uma variante do algoritmo de assinatura digital de Elgamal, especificado pelo *NIST* no documento de Padrão de Assinatura Digital (2013), que define o *DSA* para assinatura digital e SHA-1 para função de resumo. A grande diferença entre o *DSA* e o esquema de Elgamal é a questão da segurança, o tamanho da assinatura e ataques que ameaçariam o esquema de Elgamal não se aplicam ao *DSA*. O *DSA* apenas gera assinaturas e não garante a confidencialidade da mensagem.

A etapa de geração de chaves para o *DSA* é a seguinte para os parâmetros L e N (tamanho em *bits* dos números primos utilizados):

- Escolher um primo p tal que $2^{L-1} < p < 2^L$.
- Escolher um divisor primo q do número $(p-1)$, tal que $2^{N-1} < q < 2^N$.
- Escolher um gerador g de um grupo de ordem q em $GF(p)*$, onde $1 < g < p$.
- Escolher um número, que precisa ficar em segredo, aleatório (ou pseudoaleatório) x, tal que $0 < x < q$
- Calcular $y \equiv g^x \mod p$.
- Chave pública Pk é (p,q,g,y). Chave privada Sk é x.

Para gerar assinatura, o *DSA* realiza os seguintes passos:

- Escolher um número aleatório secreto k, onde $0 < k < q$.
- Calcular $r \equiv (g^k \mod p) \mod q$.
- Calcular $s \equiv [SHA-1(M) + x \cdot r]k^{-1} \mod q$.
- Assinatura da mensagem M é (r,s).

Para verificar a assinatura (r,s), deve-se realizar os seguintes passos:

- Calcular $w \equiv s^{-1} \mod q$.
- Calcular $u_1 \equiv w \cdot SHA-1(M) \mod q$ e $u_2 = r \cdot w \mod q$.

- Calcular $v \equiv (g^{u_1} \cdot y^{u_2} \mod p) \mod q$.
- Se $v \equiv r \mod q$, a assinatura é válida. Se $v \not\equiv r \mod q$, a assinatura é inválida.

O documento de especificação do *NIST*, define os valores para os parâmetros L e N em quatro grupos.

$$L = 1024, N = 160$$
$$L = 2048, N = 224$$
$$L = 2048, N = 256$$
$$L = 3072, N = 256$$

2.2.3 Elliptic Curve Digital Signature Algorithm

O esquema de assinatura digital de curvas elípticas tem uma grande vantagem em relação a outros esquemas como o RSA e o *DSA* pelo fato de os ataques à criptosistemas de curvas elípticas serem consideravelmente mais fracos (PAAR; PELZL, 2009). Esquemas de assinaturas usando curvas elípticas podem ter tamanhos de assinatura e de chaves bem menor em relação a outros esquemas, com o mesmo nível de segurança. Consequentemente, ao usar chaves e assinaturas de tamanhos melhores, esse esquema de assinatura de curvas elípticas tem um ótimo desempenho. Por motivos assim, o *ECDSA* foi padronizado nos Estados Unidos pelo *American National Standards Institute* / Instituto Nacional Americano de Padrões (*ANSI*) (1999).

As etapas do *ECDSA* são relativamente parecidas com as etapas do esquema *DSA*. Entretanto, o problema do logaritmo discreto, que é a base da sua segurança, é construído utilizando um grupo de curvas elípticas. Assim, a aritmética para computar uma assinatura no *ECDSA* é inteiramente diferente da utilizada para o DSA. O padrão *ECDSA* é definido para curvas elípticas sobre o corpo de primos $\mathbb{Z}_p$ e o corpo de *Galois* $GF(2^m)$.

A etapa para geração das chaves do *ECDSA* funciona da seguinte forma:

1. Usar uma curva elíptica E com
   - modulo p.
   - dois coeficientes a e b.
   - um ponto A que gera um grupo cíclico de ordem q de primos.
2. Escolher um número inteiro aleatório d com $0 < d < q$.
3. Calcular $B = dA$.
4. A chave pública Pk é (p, a, b, q, A, B), e a chave privada Sk é d.

Para gerar assinatura da mensagem M, o *ECDSA* realiza os seguintes passos:

1. Escolher um número aleatório secreto k, onde $0 < k < q$.
2. Calcular $R = kA$.
3. Faça $r = M_R$.
4. Calcular $s \equiv (h(M) + d \cdot r)k^{-1} \mod q$. ($h(M)$ é o resumo da mensagem)
5. Assinatura da mensagem M é (r,s).

Para verificar a assinatura (r,s), deve-se realizar os seguintes passos:

1. Calcular $w \equiv s^{-1} \mod q$.
2. Calcular $u_1 \equiv w \cdot h(M) \mod q$ e $u_2 = r \cdot w \mod q$.
3. Calcular $P = u_1 A + u_2 B$.
4. Se $x_P \equiv r \mod q$, a assinatura é válida. Se $x_P \not\equiv r \mod q$, a assinatura é inválida.

Encontrar uma curva elíptica com boas propriedades criptográficas não é uma tarefa fácil. Na prática, as curvas padronizadas, como as propostas pelo *NIST*, é que são frequentemente utilizadas.

Dado que os parâmetros da curva elíptica são escolhidos corretamente, o principal ataque analítico contra *ECDSA* tenta resolver o problema do logaritmo discreto da curva elíptica. Se um invasor pudesse fazer isso, ele poderia calcular a chave privada d.

2.3 Algoritmo de Shor

O algoritmo quântico probabilístico para fatoração de inteiros grandes e cálculo de logaritmos discretos em corpos finitos em tempo hábil proposto por Peter W. Shor (1994) é tido como o primeiro algoritmo quântico relevante tanto na prática, quanto na eficiência. Dado um inteiro qualquer n, com no mínimo dois divisores diferentes e primos, o algoritmo de Shor consegue computar um divisor não óbvio de n em tempo polinomial. Assim, trazendo novos problemas à criptografia atual, já que a segurança dos esquemas tradicionais de criptografia assimétrica mencionadas anteriormente é embasada exatamente nesses problemas ou de alguma forma relacionada a eles. Desse modo, a segurança dessas técnicas fica dependente da inexistência de um computador quântico. Isso sugere que uma informação criptografada atualmente, poderá não ser mais segura com a existência dos computadores quânticos.

Logo, com o surgimento de computadores quânticos, será necessário o uso de outros criptossistemas que sejam baseados em problemas da computação que sejam resistentes ao algoritmo de Shor e aos computadores quânticos.

2.4 Algoritmo de Grover

O algoritmo de Grover (1996) é um algoritmo quântico de busca, que tira proveito do paralelismo do computador quântico para encontrar soluções para o problema de busca. O algoritmo de Grover diminui o campo de busca para um ataque de força bruta de 2^N para $2^{N/2}$, fazendo com que o número de *bits* de segurança do campo seja reduzido pela metade. O algoritmo de Grover permite um *speedup* quadrático ao ataque de força bruta; isto é, se eram necessárias 2^{128} interações para quebrar um algoritmo simétrico com chave de 128-*bits* esse número é reduzido para 2^{64} com o uso do algoritmo de Grover. Sempre é possível fazer um ataque de força bruta a um algoritmo simétrico (2009).

Dessa forma, para que os algoritmos simétricos continuem seguros na computação quântica, deve-se dobrar o número de *bits* da chave tal como o número de *bits* de saída da função resumo.

2.5 Assinaturas Digitais Pós-Quânticas

Os esquemas de assinatura digital apresentados anteriormente têm sua segurança baseada na complexidade de fatorar números inteiros ou calcular logaritmos discretos. Shor (1994) propôs um algoritmo quântico para fatorar inteiros e calcular logaritmos discretos em tempo polinomial, usando operações simples em um computador quântico. O surgimento desses computadores quânticos poderão causar um grande problema para a segurança desses esquemas criptográficos usados atualmente, pois com a implementação do algoritmo de Shor, poderá ser possível quebrar a segurança de esquemas de assinatura digital como os citados acima.

Porém, existem outros problemas computacionais que são considerados resistentes ao algoritmo de Shor e a computadores quânticos. A partir desses problemas, foram criados novos esquemas criptográficos também resistentes a computadores quânticos. Esses esquemas passaram a ser chamados de criptossistemas pós-quânticos. Esquemas baseados em reticulados, baseado em provas de conhecimentos Zero não interativas, sistemas multivariados quadráticos ($\mathscr{MQ}$) e baseado em funções de resumo, são alguns dos novos esquemas pós-quânticos.

2.5.1 Assinatura Digital baseada em funções de resumo

Esquemas baseados em funções de resumo ficaram populares depois do aparecimento do trabalho de Merkle (1979). Em seu trabalho, Merkle, propôs um esquema de assinatura

baseado no esquema de assinatura *one-time* proposto por Lamport e Diffie em 1979. A segurança de ambos os esquemas vem das propriedades da função de resumo utilizada. O esquema de Merkle é viável para utilização prática, pois não há como aplicar o algoritmo de Shor. Para o algoritmo de Grover, basta dobrar a quantidade de *bits* das chaves e da função de resumo. Uma grande falha de esquemas *one-time*, é que uma chave privada só pode gerar uma única assinatura.

Dos algoritmos selecionados para a segunda fase da competição do *NIST* de 2017, o SPHINCS$^+$ é baseado em funções de resumo.

2.5.2 Assinatura Digital baseada em sistemas multivariados

Os esquemas baseados em sistemas multivariados fazem parte dos esquemas de assinatura digital considerados resistentes aos computadores quânticos. Esse esquema baseado em sistemas multivariados, tem sua segurança baseada na complexidade de resolver sistemas de equações não-lineares multivariadas sobre corpos finitos. Na maioria dos casos, são sistemas de equações multivariadas quadráticas. Patrin (1997) provou que o problema de resolver de sistemas de equações multivariada quadráticas, também chamado de Problema $\mathscr{MQ}$, faz parte do conjunto de problemas NP-difícil. Os estudos com relação a esquemas que usam esse problema tem se desenvolvido com mais enfoque nos últimos anos e, por isso, acabaram percebendo que alguns dos esquemas que já existiam, não eram tão seguros quanto se pensava, enquanto outros foram reanalisados e permaneceram sendo considerados seguros.

Um dos esquemas de assinatura baseado no Problema $\mathscr{MQ}$ mais conhecido, foi proposto por Patarin (1999) e, é chamado de *Unbalanced Oil and Vinegar*. Essa nomenclatura se dá pelo fato das variáveis estarem divididas em duas classes, a classe azeite (*Oil*) onde as variáveis não ocorrem juntas em termos quadráticos, e a classe vinagre (*Vinegar*) onde as variáveis já são conhecidas ou são escolhidas de forma aleatória. Essa divisão permite reduzir o sistema quadrático para um sistema linear na classe dos azeites.

Dos algoritmos selecionados para a segunda fase da competição do *NIST*, o GeMSS, o LUOV, o MQDSS e o Rainbow são baseados em sistemas multivariados.

2.5.3 Criptografia baseada em códigos

O primeiro algoritmo criado utilizando códigos corretores de erros como sua base foi o algoritmo proposto por McElice (1978) que têm o mesmo nome do seu inventor. Esse algoritmo é considerado, principalmente, como uma função de mão única com alçapão (*trapdoor*

one-way functions). Esse tipo de função pode ser transformada de diversas maneiras para criar um criptossistema. O esquema de McElice voltou a ser analisado nos últimos anos por sua velocidade de processamento, mesmo sendo um esquema tão antigo.

Um dos códigos corretos de erros que é mais frequentemente utilizado pelo algoritmo de McElice é a família de códigos de Goppa, pois tal família de códigos tem se mostrado resistentes a ataques de criptoanálise.

2.5.4 Criptografia baseada em reticulados

Ajtai (1996) deu início ao uso de reticulados em criptografia. Ajtai, em seu trabalho, mostrou que o problema de encontrar vetores pequenos em uma classe aleatória de reticulados serve como premissa na demostração da existência de função de mão única, por exemplo, funções de resumo. A criptografia baseada em reticulados é bastante visada por suas implementações serem eficientes e razoavelmente de fácil compreensão, além de ter sua segurança comprovada por demonstrações construídas em relação ao pior caso de certos problemas.

Os esquemas de criptografia baseada em reticulados podem ser particionados em dois grupos:

- (i) Esquemas com sua segurança demonstrada, por exemplo, o esquema de Ajtai, mas que não tem uma implementação eficiente, sendo assim, tendo uma complexidade alta, sendo maior até que esquemas tradicionais;
- (ii) Esquemas com implementação considerada eficiente, porém sem sua segurança, como por exemplo o esquema NTRU.

Dos algoritmos selecionados para a segunda fase da competição do *NIST*, o CRYSTALS-DILITHIUM, o FALCON e o qTESLA são baseados em reticulados.

3 ESQUEMAS DE MERKLE, LEIGHTON E MICALI E SPHINCS$^+$

Neste capítulo são descritos os conceitos dos algoritmos alvos deste trabalho. Inicialmente, será abordado os esquemas de *One-Time Signature* (*OTS*), que são a base para os esquemas de Merkle, Leighton e Micali e SPHINCS$^+$. Serão apresentados os seguintes esquemas *OTS*: esquema de Lamport e esquema de Winternitz. Depois, serão apresentados o esquema de Merkle, esquema de Merkle estendido, esquema de Leighton e Micali e SPHINCS$^+$.

3.1 Esquemas de Assinatura One-Time

Um esquema de assinatura *one-time* (*OTS*) é um esquema de assinatura digital baseado em função de resumo que emprega um par de chaves distinto para gerar a assinatura de uma mensagem. A segurança de esquemas desse tipo só é válida se uma dada chave privada só é usada para assinar apenas uma única mensagem. Lamport (1979) foi o primeiro a apresentar os esquemas de assinatura *one-time*. Outro quesito para a segurança desses esquemas, é a função de resumo que o esquema utiliza. Caso a função de resumo utilizada passe a ser insegura, o esquema ficará inseguro também, mas isso não é um grande problema já que a função de resumo pode facilmente ser substituída por outra função de resumo segura.

3.1.1 Esquema de Lamport-Diffie

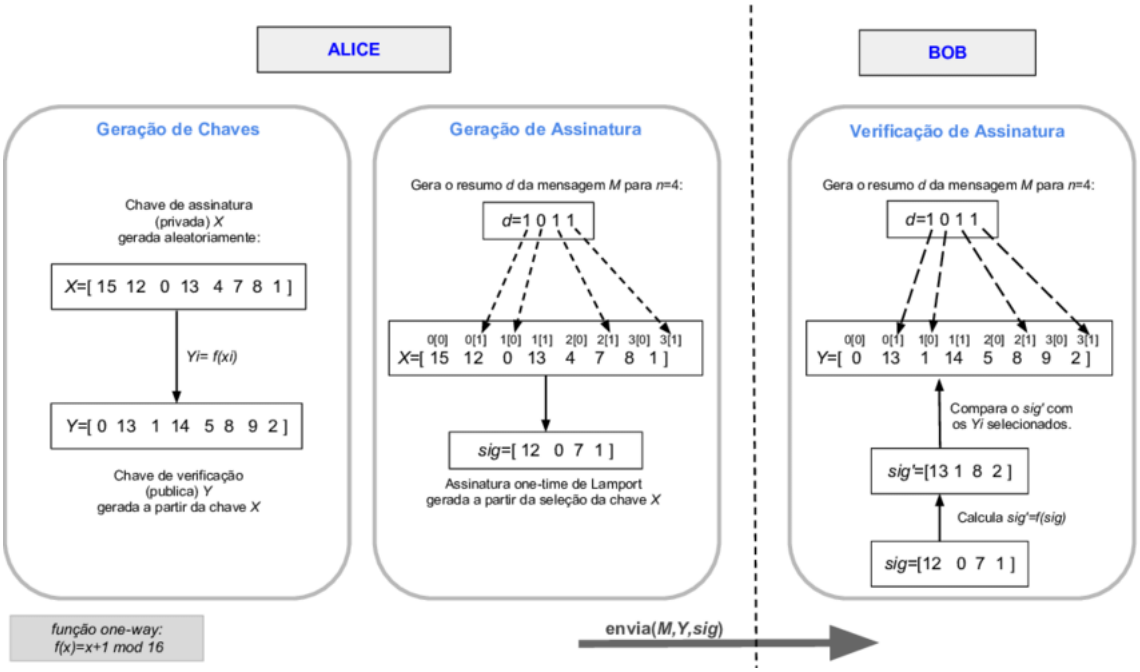
O esquema de Lamport-Diffie foi proposto por Leslie Lamport (1979), com o apoio de Whitfield Diffie, que apresentou um problema que foi o ponto de partida para a criação do esquema. Ele usa duas funções básicas: uma função unidirecional f e uma função de resumo h, com um parâmetro de segurança para ambas as funções. O esquema utiliza uma chave de assinatura e uma chave de verificação, ambas com o mesmo tamanho em *bits*; e assinaturas diferentes a cada troca de mensagem.

A chave privada do esquema é escolhida de forma aleatória e consiste de cadeias binárias de mesmo tamanho. A chave pública de verificação é o resultado da aplicação da função unidirecional f sobre a chave privada.

Após a criação das chaves, para gerar a assinatura, é aplicada a função de resumo h sobre uma mensagem qualquer M, gerando o valor de resumo, representado por uma cadeia de *bits*. A partir desse valor de resumo e da chave privada é gerada a assinatura, de forma que cada bit do valor de resumo indica qual posição da chave privada será usada para compor a assinatura.

Para verificar a assinatura da mensagem M, primeiro calcula-se o resumo da mensagem. Então, aplica-se a função unidirecional f para cada elemento da assinatura e após, compara-se com a chave de verificação. Se os elementos forem iguais, então a assinatura é válida, caso contrário é inválida. A Figura 4 mostra um exemplo simples de todos os 3 passos.

Figura 4 – Esquema de Lamport-Diffie



Fonte – Barreto *et al.* (2013)

O grande problema desse esquema, por ser um esquemas *one-time*, é que novas chaves precisam ser geradas a cada nova mensagem a ser assinada, pois se a chave for mantida, alguém poderia gerar várias mensagens para autenticação, com o propósito de descobrir qual a chave privada.

3.1.2 Esquema de Winternitz

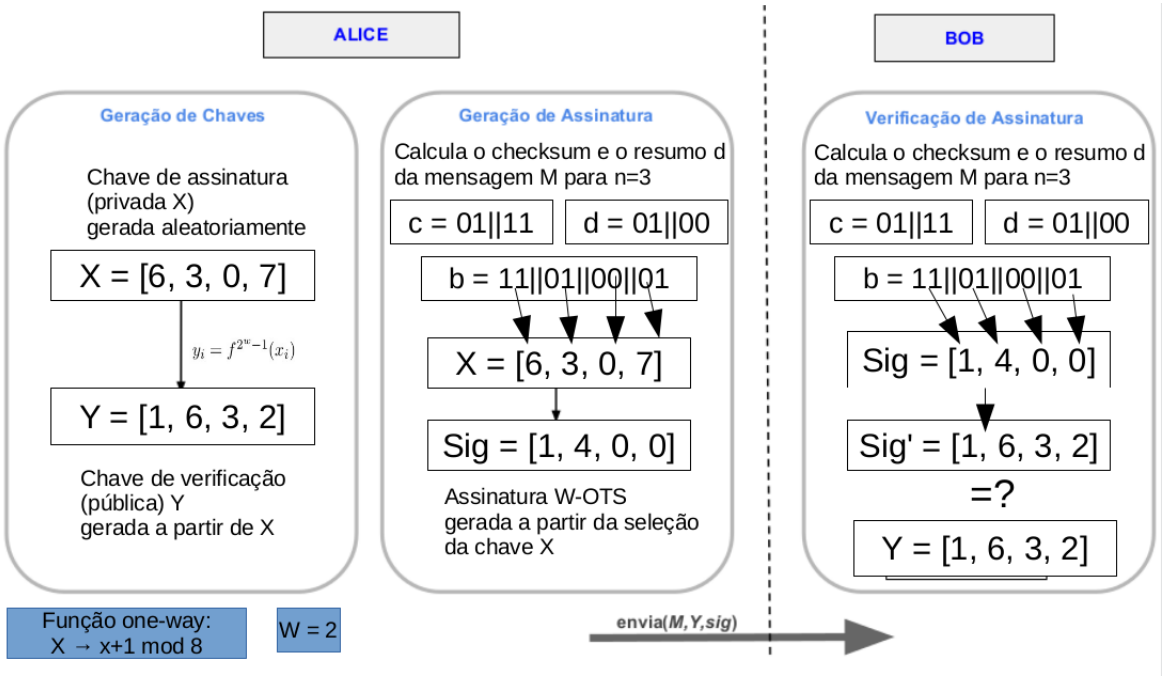
Winternitz propôs um avanço no esquema de assinatura de Lamport, introduzindo um parâmetro w, chamado parâmetro de Winternitz, que permite a criação de chaves e assinaturas menores, mas com uma diminuição no desempenho do esquema, já que serão feitas mais chamadas a função unidirecional que o esquema usa. Esse esquema foi publicado por Merkle (1979).

Este esquema possibilita assinar diversos *bits* ao mesmo tempo em blocos distintos

da assinatura, onde o parâmetro de Winternitz indica o número de *bits* que podem ser assinados ao mesmo tempo. Quanto maior for o tamanho indicado pelo parâmetro, menor será a assinatura e maior será o tempo de assinatura e verificação da assinatura.

O esquema de Winternitz, assim como o esquema de Lamport, utiliza uma função unidirecional f e uma função de resumo h, também com um parâmetro de segurança para ambas as funções. A Figura 5 mostra um exemplo simples do funcionamento do esquema de Winternitz, depois será feita uma explicação um pouco mais detalhada do processo inteiro.

Figura 5 – Esquema de Winternitz



Fonte – Nossa autoria

Para a geração das chaves, deve-se escolher um valor para o parâmetro w, então a chave privada é escolhida de forma aleatória. O tamanho da chave é obtido através de um cálculo utilizando o parâmetro w. A chave pública de verificação é gerada aplicando a função unidirecional f a cada elemento da chave privada $2^w - 1$ vezes.

Para criar uma assinatura, primeiro é aplicada a função de resumo h sobre a mensagem M. Se for preciso, adicionamos zeros a esquerda do valor de resumo, de forma que o valor de resumo seja divisível pelo parâmetro w. Então o valor de resumo é dividido em blocos de tamanho w. Adicionalmente, uma soma de verificação é calculada. Se for preciso, são adicionados zeros à esquerda do valor da soma de verificação de modo que o novo valor da soma de verificação seja divisível por w. Então, o novo valor da soma de verificação pode ser dividido

em blocos, também, de comprimento w. Calculamos a assinatura, como a aplicação da função unidirecional f em cada bloco da chave privada um valor (fixo) de vezes, que é computado usando o resumo da mensagem concatenado com a soma de resumo.

Para verificar a assinatura da mensagem M, primeiro precisamos calcular a concatenação dos valores de resumo e da soma de verificação do mesmo modo que durante a geração da assinatura. Depois, aplica-se a função unidirecional f sobre a assinatura $2^w - 1 - (con)$ vezes, onde *con* é o valor da concatenação entre o valor de resumo e a soma de verificação. Então, verifique se o resultado da aplicação da função f é igual a chave pública de verificação, se forem iguais a assinatura é válida, caso contrário é inválida.

3.2 Esquema de Merkle

Por causa de limitação de um par de chaves poder ser utilizado para gerar uma única assinatura, os esquemas de assinatura *one-time* não são utilizados por grande parte das aplicações práticas (BUCHMANN *et al.*, 2009). Para contornar esse problema, Merkle (1979) propôs o *Merkle Signature Scheme* / Esquema de Assinatura de Merkle (*MSS*) que permite assinar um grande número de mensagens. Esta estratégia permite que a validação de todas as assinaturas seja feita utilizando apenas uma chave pública, a raiz de uma árvore binária, e não várias chaves de verificações, uma para cada mensagem como ocorreria em esquemas *one-time*. O esquema de Merkle utiliza uma função de resumo g e um esquema de assinatura *one-time*, desde de que, o esquema escolhido seja seguro.

O esquema de Merkle monta uma árvore binária na qual as folhas dessa árvore estão associadas a um par de chave; uma chave de assinatura e uma chave de verificação *one-time*, onde o valor de cada uma dessas folhas armazena o valor de resumo de uma chave de verificação *one-time*. A chave pública do esquema de Merkle tem seu valor armazenado na raiz da árvore, para que essa chave pública seja gerada, é necessário primeiro gerar cada par de chaves para as folhas da árvore. Uma árvore com altura h e 2^h folhas terá 2^h pares de chaves públicas e privadas *one-time*.

No esquema de Merkle, uma chave privada do esquema é, na verdade, um conjunto de chaves privadas *one-time*, pois cada chave privada estará associada a uma folha da árvore que é gerada. Um ótimo meio para gerar as chaves privadas é utilizar uma função pseudoaleatória, pois com seu uso, ao invés de se armazenar todas as chaves geradas, pode-se armazenar apenas a semente usada na função, sendo assim, as chaves podem ser geradas novamente sempre que

for necessário a partir da semente armazenada. Como cada chave está associada a uma folha da árvore, cada uma delas tem que gerar seu par de chaves usando a função pseudoaleatória, seu índice na árvore e a semente.

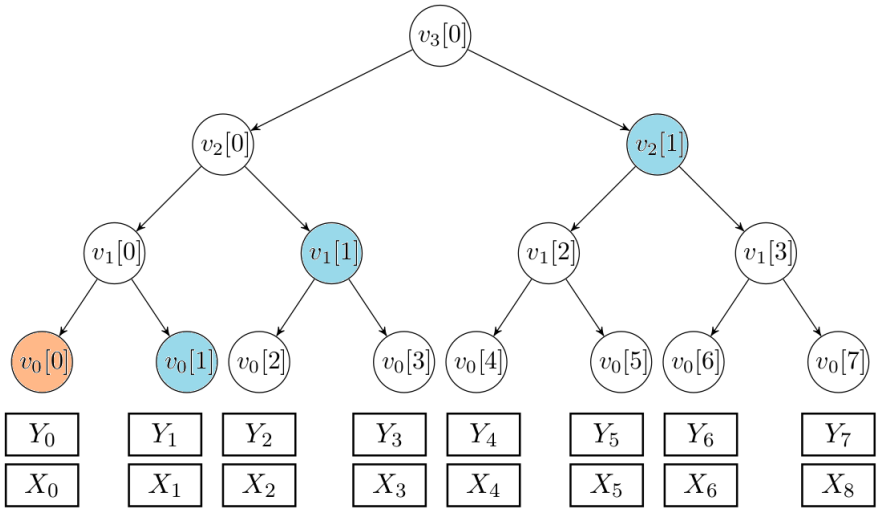
Após a geração de todos os pares de chaves para cada nó folha da árvore, o esquema poderá gerar sua chave pública. Durante a geração de cada par de chave, o nó folha recebe o valor de resumo sobre a chave de verificação. Os nós internos irão receber o valor de resumo da concatenação do valor presente em seus filhos, esquerdo e direito. Esse processo é feito até que se chegue na raiz da árvore. Quando a raiz da árvore tem seu valor calculado, teremos a chave pública do esquema de Merkle.

Para mostrar que as assinaturas são válidas, é preciso mostrar a autenticidade das mesmas. Para isso, faz-se uso do caminho de autenticação de assinatura. Esse caminho de autenticação é o conjunto dos valores dos nós irmãos em cada altura que ligam a folha, chave de assinatura, até a raiz da árvore do esquema de Merkle, chave pública. O uso desse caminho de autenticação diminui a necessidade de enviar a árvore toda e envia apenas um pequeno conjunto de nós para realizar a verificação.

Com uma árvore de altura h, o esquema de Merkle é capaz de criar 2^h assinaturas. A geração de uma assinatura é feita em dois passos: primeiro, a assinatura do resumo da mensagem M é criada utilizando o esquema *one-time* escolhido com uma chave privada, associada a uma folha da árvore. No próximo passo, o caminho de autenticação para a próxima folha da árvore é preparado. Tal caminho é computado de forma ágil, pois somente computa os nós do caminho de autenticação que mudam na próxima assinatura. A Figura 6 mostra um caminho de autenticação, os nós na cor azul, para o nó folha na cor laranja.

Para verificar uma assinatura, precisam ser realizadas duas fases. Na primeira fase, a assinatura é verificada usando a chave de verificação *one-time* e o respectivo esquema *one-time* escolhido. Na segunda fase, é preciso validar a chave pública da árvore de Merkle, por isso, é necessário o caminho de autenticação, para ir construindo o caminho do nó folha, que contém a chave de verificação *one-time*, até a raiz. Se o valor encontrado pelo caminho de autenticação for igual ao valor da chave pública, que é conhecida, então a assinatura é válida. Caso o valor encontrado seja diferente, a assinatura é inválida.

Figura 6 – Representação da árvore de Merkle com o caminho de autenticação



Fonte – Própria autoria

3.3 Esquema de Merkle Estendido

O *XMSS* (2011), assim como o *MSS*, é um método que consegue assinar um número relativamente grande, mas fixo de mensagens. Ele faz uso do esquema WOTS$^+$, uma variante do esquema de Winternitz. Uma característica para que o *XMSS* seja seguro, é que o esquema não requer que a função de resumo seja resistente à colisão (BUTIN *et al.*, 2017). Foi evidenciado que quando o *XMSS* é instanciado com duas famílias de funções o esquema é considerado eficiente e *forward secure*[1]. Essas duas famílias de funções são uma família de funções de resumo, que deve ser resistente a segunda pré-imagem e uma família de funções pseudoaleatórias.

O *XMSS* tem os seguintes parâmetros: h, a altura da árvore; n, o tamanho da saída da função de resumo em bits; $w(w > 1)$, o parâmetro de Winternitz. Considere também h, uma função de resumo e f, uma função pseudoaleatória.

Para gerar as chaves privadas do *XMSS*, usa-se a função pseudoaleatória f para diminuir custos de armazenamento. Para gerar a chave pública do *XMSS*, é necessário gerar uma árvore binária do *XMSS*. Cada nó folha da árvore do *XMSS* é uma Ltree, uma árvore binária não balanceada, onde as folhas dessa árvore são chaves públicas de verificação do esquema WOTS$^+$. Assim, como no Esquema de Merkle, a chave pública do *XMSS* é raiz da árvore binária gerada. Durante a geração da árvore, são geradas 2^h chaves de assinatura e verificação do esquema WOTS$^+$.

[1] Recurso utilizado para proteger o conteúdo de mensagens criptografadas e assinaturas digitais antigas contra eventuais vazamentos ou comprometimentos futuros de chaves.

Para gerar uma assinatura do *XMSS* é necessário realizar duas etapas: a assinatura do resumo da mensagem usando uma chave privada do WOTS$^+$ e o caminho de autenticação para a folha contendo a chave utilizada para assinar o resumo da mensagem.

A verificação de uma uma assinatura do *XMSS* é exatamente igual a uma verificação de assinatura do Esquema de Merkle. O *XMSS* como um todo é basicamente igual ao Esquema de Merkle, a diferença principal é a utilização de uma máscara de *bits* que é usada durante a construção da árvore, e no momento da verificação. Devido a existência dessa máscara de *bits*, o *XMSS* não requer que a função de resumo seja resiste a colisão, ela deve ser resistente apenas a segunda pré-imagem.

3.4 Esquema de Leighton e Micali

O esquema *Leighton-Micali Signature* / Assinatura de Leighton e Micali (*LMS*) (MCGREW *et al.*, 2016) pode assinar uma alta quantidade, porém fixa, de mensagens. Esse esquema, assim como o de Merkle, usa um esquema de assinatura *one-time* e uma função de resumo. Cada par de chave pública e privada do esquema *LMS* é uma folha de uma árvore binária, assim como a árvore de Merkle.

O esquema *LMS* tem dois parâmetros principais, h a altura da árvore e n o número de *bytes* de cada nó. A árvore gerada do *LMS* tem 2^h folhas. Cada folha da árvore contém o valor da chave pública. A chave pública do esquema *LMS* é o valor da raiz da árvore, da mesma forma que o Esquema de Merkle.

Uma chave privada do *LMS* é representada por 2^h chaves privadas *one-time* e o do próximo nó folha que ainda não foi usado. A chave pública do esquema *LMS*, é associada com as chaves públicas e privadas do esquema *one-time* utilizado. Cada folha da árvore do *LMS* recebe uma chave pública.

A assinatura do esquema *LMS* é constituída de um código indicando os parâmetros do esquema, do índice da folha associado com a assinatura, da assinatura, e do caminho de autenticação.

Por também conter um caminho de autenticação na assinatura, a verificação da assinatura do *LMS* funciona de forma similar a verificação do Esquema de Merkle e o *XMSS*, primeiro verificando a assinatura e validando a assinatura pelo caminho de autenticação. A assinatura é válida se o valor da raiz for igual a chave pública do esquema *LMS*, se não for a assinatura é inválida.

3.5 SPHINCS$^+$

Para falar do SPHINCS$^+$, devemos primeiro introduzir seu predecessor, o SPHINCS, pois analisando de forma simplificada, o SPHINCS$^+$ funciona de maneira similar ao SPHINCS.

O SPHINCS (2015) foi projetado como um esquema de assinatura digital *stateless* baseado em função de resumo e resistente contra ataques que utilizam um computador quântico. Grande parte dos esquemas de assinatura baseados em funções de resumo são *statefull*, por exemplo o *XMSS*, pois necessitam guardar certas informações sobre seu estado depois de cada assinatura. Por ser um esquema *stateless*, o SPHINCS tem sua segurança melhorada, pelo fato de não precisar guardar tais informações. O SPHINCS utiliza muitos dos componentes do *XMSS*, porém com tamanhos bem maiores de chaves e assinaturas para eleminar a questão do seu estado.

Bernstein *et al.*, dizem que o SPHINCS$^+$ utiliza um *hypertree* para autenticar um grande número de pares de chaves de assinatura *few-time*, que são esquemas que permitem um par de chaves produzir uma pequena quantidade de assinaturas. Para assinar uma mensagem, um par de chave *few-time* é escolhido de forma pseudoaleatória. A assinatura da mensagem se dá pelas informações de autenticação sobre tal par de chaves e a assinatura *few-time*.

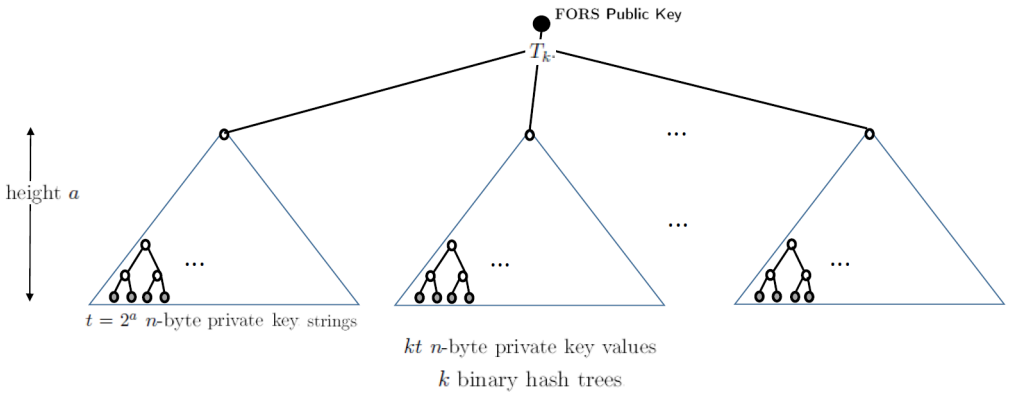
Uma *hypertree* é dita ser uma árvore de assinatura *many-time* baseada em função de resumo, por exemplo, a árvore de Merkle utilizada pelo esquemas de assinatura de Merkle. Esses esquemas de assinatura *many-time* permitem um par de chaves assinar um número fixo de mensagens. As informações de autenticação sobre um par de chaves *few-time* se resume nas assinaturas *many-time* que formam um caminho partindo do par de chaves *few-time* até o topo da árvore *many-time*.

A árvore *many-time* que o SPHINCS$^+$ utiliza é essencialmente uma árvore de Merkle clássica. Mais precisamente, a *hypertree* utilizada é uma variante da versão multi-árvore do Esquema de Merkle Estendido. Então, essa *hypertree* nada mais é que uma árvore em que cada nó é uma árvore do Esquema de Merkle Estendido. Os nós do topo e das camadas intermediárias são usados para assinar as chaves públicas, enquanto os nós da camada mais inferior são utilizados para assinar as mensagens. Todas as árvores em cada nó tem a mesma altura.

Mas essa *hypertree* não é utilizada para assinar as mensagens. Ela é utilizada para assinar as chaves públicas de instâncias de um esquema de assinatura *few-time* chamado *FORS*, que por questões de segurança, é necessário que a entrada para o *FORS* seja a saída de uma função de resumo. A Figura 7 apresenta o *FORS*.

A chave pública do SPHINCS$^+$ é dada pela raiz da árvore *many-time*, o valor de

Figura 7 – *FORS* e chave pública do SPHINCS$^+$



Fonte – Bernstein *et al.* (2015)

resumo da raiz da árvore de Merkle, mais uma semente pública do mesmo tamanho da raiz. A chave secreta do SPHINCS$^+$ é composta por duas partes, a primeira é o valor de uma semente secreta, a qual gera todas as chaves secretas *one-time* e *few-time*, que determinam toda à estrutura virtual de um par de chaves do SPHINCS$^+$, de maneira pseudoaleatória, e a segunda parte é uma chave PRF, que é usada para gerar deterministicamente um valor de randomização para o valor de resumo da mensagem.

A assinatura do SPHINCS$^+$ consiste de três partes: uma string aleatória, a assinatura gerada pelo esquema *few-time FORS* e pela assinatura gerada pela *hypertree*. Para gerar essa assinatura, o SPHINCS$^+$ trabalha em quatro etapas. Primeiro é gerada a string de randomização de forma pseudoaleatória. O segundo passo é gerar o valor de resumo da mensagem. A terceira etapa é assinar o valor de resumo da mensagem com um par de chaves do esquema *FORS*. E por ultimo, a chave pública do par de chaves do esquema *FORS* é assinada usando a *hypertree*.

A verificação da assinatura do SPHINCS$^+$, de forma resumida, é calcular o valor de resumo da mensagem novamente, calcular a chave pública candidata do esquema *FORS* e verificar a assinatura da *hypertree* sobre essa chave pública candidata. Se a verificação da assinatura da *hypertree* for bem sucedida significa que a chave candidata do esquema *FORS* correspondo com a chave utilizada, dessa forma, mostrando que a assinatura é válida. Se a verificação da assinatura da *hypertree* falhar, significa que a chave pública candidata do esquema *FORS* não corresponde com a chave realmente utilizada, dessa forma, mostrando que a assinatura é inválida.

4 ALGORITMOS

Neste capítulo são descritas as camadas dos algoritmos dos esquemas alvos deste trabalho, o esquema de Merkle estendido (*XMSS*) e o SPHINCS$^+$.

4.1 XMSS

Utilizamos a implementação de referência do *XMSS*, implementada por Andreas Hülsing e Joost Rijneveld. Pode ser encontrada no GitHub.

Inicialmente, o algoritmo faz definições para as funções que serão utilizadas. Depois, são definidos todos os parâmetros principais do algoritmo, como a função de resumo utilizada, o parâmetro de Winternitz, altura da árvore, entre outros. A única modificação que fizemos foi na mensagem a ser assinada, para ser a mesma em todos os testes.

Esse algoritmo, ao invés de utilizar o método normal para o *XMSS*, ele utiliza a implementação do *XMSS* versão multi-árvore (XMSS^{MT}) com apenas 1 nível, que equivale a versão normal do *XMSS*. Os motivos para a implementação feita dessa forma é que as assinaturas do *XMSS* são uma instância das assinaturas do XMSS^{MT} e os procedimentos para geração de chaves são exatamente os mesmos para ambos os algoritmos. Outra informação relevante é que por utilizar o algoritmo do XMSS^{MT}, o endereçamento para o *XMSS* é um pouco mais simplificado.

Após definir todos os parâmetros, as chaves são geradas de forma pseudoaleatória, a partir de uma semente também gerada de forma aleatória. Nesse algoritmo, ao gerar as chaves, o caminho de autenticação também é gerado para simplificar o código como um todo. O pseudo Algoritmo 1 mostra a geração de chaves do *XMSS*.

Algoritmo 1 – Geração do par de chave do XMSS

```
1 Entrada: Sem entrada
2 Saída: Chave privada SK do XMSS, Chave publica PK do XMSS
3
4   idx = 0;
5   for ( i = 0; i < 2^h; i++ ) {
6     wots_sk[i] = WOTS_genSK();
7   }
```

```
8   initialize SK_PRF with a uniformly random n-byte string;
9   setSK_PRF(SK, SK_PRF);
10
11  initialize SEED with a uniformly random n-byte string;
12  setSEED(SK, SEED);
13  setWOTS_SK(SK, wots_sk));
14  ADRS = toByte(0, 32);
15  root = treeHash(SK, 0, h, ADRS);
16
17  SK = idx || wots_sk || SK_PRF || root || SEED;
18  PK = OID || root || SEED;
19  return (SK || PK);
```

Fonte – Hülsing *et al.* (2018)

Ao assinar a mensagem, o algoritmo utiliza o índice atual da chave, computa um randomizador para o resumo, computa o resumo da mensagem. Após essas preparações, gera a semente para criar o par de chaves do esquema de Winternitz, computa a assinatura do esquema de Winternitz e o caminho de autenticação para a folha utilizada. Os pseudos Algoritmos 2 e 3 mostram a assinatura da mensagem do *XMSS*.

Algoritmo 2 – Assinatura do XMSS

```
1  Entrada: Mensagem M, Chave privada SK do XMSS
2  Saída: Chave privada SK atualizada, assinatura Sig do XMSS
3
4      idx_sig = getIdx(SK);
5      setIdx(SK, idx_sig + 1);
6      ADRS = toByte(0, 32);
7      byte[n] r = PRF(getSK_PRF(SK), toByte(idx_sig, 32));
8      byte[n] M' = H_msg(r || getRoot(SK) || (toByte(idx_sig,
           n)), M);
9      Sig = idx_sig || r || treeSig(M', SK, idx_sig, ADRS);
10     return (SK || Sig);
```

Fonte – Hülsing *et al.* (2018)

Algoritmo 3 – Assinatura do WOTS+ e caminho de autenticação - (treeSig)

```
Entrada: Mensagem M', Chave privada SK, índice idx_sig da
   assinatura, ADRS
Saída: Assinatura sig_ots do WOTS+ e o caminho de autentica
   ção

    for ( j = 0; j < h; j++ ) {
      k = floor(i / (2^j)) XOR 1;
      auth[j] = treeHash(SK, k * 2^j, j, ADRS);
    }
    ADRS.setType(0);   // Type = OTS hash address
    ADRS.setOTSAddress(idx_sig);
    sig_ots = WOTS_sign(getWOTS_SK(SK, idx_sig), M',
       getSEED(SK), ADRS);
    Sig = sig_ots || auth;
    return Sig;
```

Fonte – Hülsing *et al.* (2018)

Ao verificar a assinatura, o algoritmo computa o resumo da mensagem. Então computa o nó folha usando a chave pública do esquema de Winternitz e, por fim, o nó raiz. Feito isso, o algoritmo verifica se o nó raiz encontrado é igual ao nó raiz da chave pública. O pseudo Algoritmo 4 mostra a verificação da assinatura do *XMSS*.

Algoritmo 4 – Verificação da assinatura do XMSS

```
Entrada: Assinatura Sig do XMSS, mensagem M, Chave publica
   PK do XMSS
Saída: Booleano

    ADRS = toByte(0, 32);
    byte[n] M' = H_msg(r || getRoot(PK) || (toByte(idx_sig,
```

```
        n)), M);
6
7       ADRS.setType(0);   // Type = OTS hash address
8       ADRS.setOTSAddress(idx_sig);
9       pk_ots = WOTS_pkFromSig(sig_ots, M', SEED, ADRS);
10      ADRS.setType(1);   // Type = L-tree address
11      ADRS.setLTreeAddress(idx_sig);
12      byte[n][2] node;
13      node[0] = ltree(pk_ots, SEED, ADRS);
14      ADRS.setType(2);   // Type = hash tree address
15      ADRS.setTreeIndex(idx_sig);
16      for ( k = 0; k < h; k++ ) {
17          ADRS.setTreeHeight(k);
18          if ( (floor(idx_sig / (2^k)) % 2) == 0 ) {
19              ADRS.setTreeIndex(ADRS.getTreeIndex() / 2);
20              node[1] = RAND_HASH(node[0], auth[k], SEED,
                    ADRS);
21          } else {
22              ADRS.setTreeIndex((ADRS.getTreeIndex() - 1) /
                    2);
23              node[1] = RAND_HASH(auth[k], node[0], SEED,
                    ADRS);
24          }
25          node[0] = node[1];
26      }
27
28      if ( node[0] == getRoot(PK) ) {
29          return true;
30      } else {
31          return false;
32      }
```

Fonte – Hülsing *et al.* (2018)

4.2 SPHINCS$^+$

Utilizamos a implementação de referência do SPHINCS$^+$, implementada por Andreas Hülsing e Joost Rijneveld. Pode ser encontrada no site para o SPHINCS$^+$.

As chaves são geradas de forma pseudoaleatória, a partir de uma semente também gerada de forma aleatória. Nesse algoritmo, ao gerar as chaves o caminho de autenticação também é gerado para simplificar o código como um todo. O pseudo Algoritmo 5 mostra a geração de chaves do SPHINCS$^+$.

Algoritmo 5 – Geração do par de chave do SPHINCS+

```
1 Entrada: Sem entrada
2 Saída: Par de chaves (SK, PK) do SPHINCS+
3 spx_keygen( ){
4     SK.seed = sec_rand(n);
5     SK.prf = sec_rand(n);
6     PK.seed = sec_rand(n);
7     PK.root = ht_PKgen(SK.seed, PK.seed);
8     return ((SK.seed, SK.prf, PK.seed, PK.root), (PK.seed,
          PK.root));
9 }
```

Fonte – Bernstein *et al.* (2017)

Ao assinar a mensagem, o algoritmo computa um randomizador para o resumo e computa o resumo da mensagem. Após essas preparações, gera a assinatura do resumo da mensagem usando *FORS*, computa a assinatura do esquema de Winternitz, o caminho de autenticação para a folha utilizada e atualiza o índice para a próxima camada. O pseudo Algoritmo 6 mostra a assinatura da mensagem do SPHINCS$^+$.

Algoritmo 6 – Assinatura do SPHINCS+

```
1 Entrada: Mensagem M, chave privada SK (SK.seed, SK.prf, PK.
     seed, PK.root)
```

```
Saída: Assinatura SIG do SPHINCS+
spx_sign(M, SK){
    // init
    ADRS = toByte(0, 32);

    // generate randomizer
    opt = toByte(0, 32);
    if(RANDOMIZE){
        opt = rand(n);
    }
    R = PRF_msg(SK.prf, opt, M);
    SIG = SIG || R;

    // compute message digest and index
    digest = H_msg(R, PK.seed, PK.root, M);
    tmp_md = first floor((ka +7)/ 8) bytes of digest;
    tmp_idx_tree = next floor((h - h/d +7)/ 8) bytes of
       digest;
    tmp_idx_leaf = next floor((h/d +7)/ 8) bytes of digest;

    md = first ka bits of tmp_md;
    idx_tree = first h - h/d bits of tmp_idx_tree;
    idx_leaf = first h/d bits of tmp_idx_leaf;

    // FORS sign
    ADRS.setLayerAddress(0);
    ADRS.setTreeAddress(idx_tree);
    ADRS.setType(FORS_TREE);
    ADRS.setKeyPairAddress(idx_leaf);

    SIG_FORS = fors_sign(md, SK.seed, PK.seed, ADRS);
    SIG = SIG || SIG_FORS;
```

```
33
34      // get FORS public key
35      PK_FORS = fors_pkFromSig(SIG_FORS, M, PK.seed, ADRS);
36
37      // sign FORS public key with HT
38      ADRS.setType(TREE);
39      SIG_HT = ht_sign(PK_FORS, SK.seed, PK.seed, idx_tree,
           idx_leaf);
40      SIG = SIG || SIG_HT;
41
42      return SIG;
43  }
```

Fonte – Bernstein *et al.* (2017)

Ao verificar a assinatura, o algoritmo computa o resumo da mensagem e o índice da folha. Para cada sub-árvore, inicialmente a raiz é a chave publica do *FORS*, mas em iterações subsequentes, é a raiz da sub-árvore abaixo da que está sendo atualmente processada. Então computa o nó folha usando a chave pública do esquema de Winternitz, computa o nó raiz e por fim atualiza o índice para a próxima camada. Feito isso, o algoritmo verifica se o nó raiz encontrado é igual ao nó raiz da chave pública. O pseudo Algoritmo 7 mostra a verificação da assinatura do SPHINCS$^+$.

Algoritmo 7 – Verificação da assinatura do SPHINCS+

```
1  Entrada: Mensagem M, assinatura SIG, chave pública PK
2  Saída: Booleano
3  spx_verify(M, SIG, PK){
4      // init
5      ADRS = toByte(0, 32);
6      R = SIG.getR();
7      SIG_FORS = SIG.getSIG_FORS();
8      SIG_HT = SIG.getSIG_HT();
9
10     // compute message digest and index
```

```
11      digest = H_msg(R, PK.seed, PK.root, M);
12      tmp_md = first floor((ka +7)/ 8) bytes of digest;
13      tmp_idx_tree = next floor((h - h/d +7)/ 8) bytes of
           digest;
14      tmp_idx_leaf = next floor((h/d +7)/ 8) bytes of digest;
15
16      md = first ka bits of tmp_md;
17      idx_tree = first h - h/d bits of tmp_idx_tree;
18      idx_leaf = first h/d bits of tmp_idx_leaf;
19
20      // compute FORS public key
21      ADRS.setLayerAddress(0);
22      ADRS.setTreeAddress(idx_tree);
23      ADRS.setType(FORS_TREE);
24      ADRS.setKeyPairAddress(idx_leaf);
25
26      PK_FORS = fors_pkFromSig(SIG_FORS, md, PK.seed, ADRS);
27
28      // verify HT signature
29      ADRS.setType(TREE);
30      return ht_verify(PK_FORS,SIG_HT,PK.seed,idx_tree,
           idx_leaf,PK.root);
31  }
```

Fonte – Bernstein *et al.* (2017)

5 RESULTADOS

Neste capítulo serão demostrados os resultados dos testes realizados com os algoritmos alvos deste trabalho, ambos os algoritmos com nível de segurança de 256 *bits*.

5.1 Parâmetros

Cada esquema tem seu próprio conjunto de parâmetros, mas são bens similares. Não foi realizada nenhuma alteração nos parâmetros principais, estão todos com os valores padrões, com exceção da mensagem a ser assinada. Foram utilizadas mensagens criadas aleatoriamente com tamanhos de 1 KB, 128 KB, 256 KB e 1024 KB. Os outros parâmetros tiveram os seguintes valores:

- parâmetro de Winternitz $w = 16$;
- tamanho da saída da função de resumo $n = 32$, que representa o parâmetro de segurança;
- a altura da árvore para o *XMSS* foi $h = 10$, mas como o SPHINCS$^+$ é um algoritmo que utiliza multi-árvore, o valor é para a *hypertree*, $h = 68$;
- o parâmetro para o número de camadas de sub árvore para o *XMSS* não deveria existir, mas como o algoritmo utilizado é o algoritmo do XMSS^{MT} o parâmetro teve o valor de $d = 1$, já que o *XMSS* é simplesmente uma instancia do XMSS^{MT} com apenas uma camada. Já para o SPHINCS$^+$, o número de camadas é $d = 17$;
- O parâmetro para a altura da FORS é exclusivo do algoritmo SPHINCS$^+$, sendo esse valor $FORS_h = 10$.

5.2 Resultados

Nesta seção são apresentados os resultados dos testes sobre a implementação de referência dos algoritmos *XMSS* e SPHINCS$^+$ na linguagem C. Os resultados foram obtidos em testes sobre os processadores Intel Core I5 7600 e Intel Core I7 8700.

A Tabela 1 mostra os tempos de execução das funções de geração de chaves, geração da assinatura e verificação da assinatura utilizadas pelos algoritmos *XMSS* com as funções de resumo SHA-256 e SHAKE e SPHINCS$^+$ com as funções de resumo SHA-256, SHAKE e HARAKA.

As Tabelas 2 e 3 mostram os tamanhos em bytes das chaves pública e privada e também o tamanho da assinatura gerada pelos algoritmos *XMSS* e SPHINCS$^+$. O tamanho da

Tabela 1 – Tempo de execução para geração das chaves dos algoritmos

Tempo de execução dos algoritmos em milissegundos (ms)					
Algoritmo	XMSS		SPHINCS$^+$		
	SHA-256	SHAKE	SHA-256	SHAKE	HARAKA
Geração de chaves	1207.71	1922.17	17.26	18.49	6.70
Geração de assinatura	2.63	3.88	377.81	399.48	257.23
Verificação de assinatura	1.47	2.14	10.57	11.63	20.62

Tabela 2 – Tamanho das chaves em bytes do XMSS e SPHINCS$^+$

Nível de segurança com $n = 32$		
Chave	XMSS	SPHINCS$^+$
PK	68	64
SK	136	128

Tabela 3 – Tamanho da assinatura em bytes do XMSS e SPHINCS$^+$

Nível de segurança com $n = 32$		
Tamanho da mensagem	XMSS	SPHINCS$^+$
1 KB	3524	50240
128 KB	133572	180288
256 KB	264644	311360
1024 KB	1051076	1097792

assinatura para o *XMSS* é $2500 + mlen$ bytes, onde *mlen* é o tamanho da mensagem em bytes. O tamanho da assinatura para o SPHINCS$^+$ é $49216 + mlen$ bytes, onde *mlen* é o tamanho da mensagem em bytes. Ambos os valores são computados a partir de somas e multiplicações dos parâmetros altura da árvore, o parâmetro de segurança *n*, o tamanho da assinatura do esquema de Winternitz e o número de camadas da sub-árvore. O SPHINCS$^+$ ainda utiliza mais um parâmetro, o tamanho da assinatura da *FORS*.

Analisando os resultados obtidos na Tabela 1, para a geração de chaves, o SPHINCS$^+$ com a função de resumo HARAKA é o algoritmo mais rápido. Em questão de geração e verificação da assinatura, o *XMSS* acaba tendo um desempenho bem melhor. Outra coisa que podemos observar na Tabela 1, o algoritmo de verificação da assinatura no SPHINCS$^+$ com a função de resumo HARAKA foi o que mostrou maior tempo de verificação.

6 CONCLUSÕES E TRABALHOS FUTUROS

Até o momento da escrita desse trabalho, o algoritmo do *XMSS* utilizado segue sendo atualizado baseado no RFC 8391 versão 12 publicado por Hülsing *et al.* (2018). Enquanto o SPHINCS$^+$ ainda segue em sua primeira versão e submetido a segunda rodada da competição do *NIST*. O algoritmo submetido para a segunda rodada introduz uma variante simples e uma robusta para cada opção de função de resumo. A variante robusta é exatamente a mesma versão da primeira submissão com todas as garantias de segurança. As variantes simples são instâncias puras do oráculo aleatório, onde essas instâncias são até 3 vezes mais rápidas que as variantes robustas ao custo de um argumento de segurança heurístico.

Este trabalho contribuiu com estudo e comparação entre um atualmente padronizado algoritmo de assinatura digital resistente a computadores quântico e um novo algoritmo candidato a padronização. Foram estudados e analisado os algoritmos *XMSS* e SPHINCS$^+$ com as funções de resumo SHA-256 e SHAKE em ambos os algoritmos assim como também o uso da função de resumo HARAKA pelo algoritmo SPHINCS$^+$.

Os resultados deste trabalho mostram que o novo candidato SPHINCS$^+$ mostrou resultados superiores, com todas as funções de resumo, para o algoritmo de geração do par de chaves em relação ao *XMSS*, entretanto, o algoritmo de geração e verificação da assinatura no *XMSS* tem um desempenho superior.

Posteriormente, em trabalhos futuros, poderia ser feita uma análise aprofundada sobre a resistência a ataques criptográficos para ambos os algoritmos.

REFERÊNCIAS

AJTAI, M. Generating hard instances of lattice problems. In: ACM. **Proceedings of the twenty-eighth annual ACM symposium on Theory of computing**. [*S. l.*], 1996. p. 99–108.

ANSI, X. 62: Public key cryptography for the financial services industry: The elliptic curve digital signature algorithm (ecdsa). **Am. Nat'l Standards Inst**, 1999.

BARKER, E. B. **Digital Signature Standard (DSS)**. [*S. l.*], 2013.

BARRETO, P.; BIASI, F. P.; DAHAB, R.; CÉSAR, J.; PEREIRA, G.; RICARDINI, J. E. Introdução à criptografia pós-quântica. **Minicursos do XIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais, SBSeg**, 2013.

BERNSTEIN, D. J.; DOBRAUNIG, C.; EICHLSEDER, M.; FLUHRER, S.; GAZDAG, S.-L.; HÜLSING, A.; KAMPANAKIS, P.; KÖLBL, S.; LANGE, T.; LAURIDSEN, M. M. *et al.* Sphincs. 2017.

BERNSTEIN, D. J.; HOPWOOD, D.; HÜLSING, A.; LANGE, T.; NIEDERHAGEN, R.; PAPACHRISTODOULOU, L.; SCHNEIDER, M.; SCHWABE, P.; WILCOX-O'HEARN, Z. Sphincs: practical stateless hash-based signatures. In: SPRINGER. **Annual International Conference on the Theory and Applications of Cryptographic Techniques**. [*S. l.*], 2015. p. 368–397.

BERTONI, G.; DAEMEN, J.; PEETERS, M.; ASSCHE, G. V. Sponge functions. In: **ECRYPT hash workshop**. [*S. l.*: *s. n.*], 2007. v. 2007, n. 9.

BERTONI, G.; DAEMEN, J.; PEETERS, M.; ASSCHE, G. V. Keccak specifications. **Submission to nist (round 2)**, Citeseer, p. 320–337, 2009.

BUCHMANN, J.; DAHMEN, E.; HÜLSING, A. Xmss-a practical forward secure signature scheme based on minimal security assumptions. **Post-Quantum Cryptography**, Springer, p. 117–129, 2011.

BUCHMANN, J.; DAHMEN, E.; SZYDLO, M. Hash-based digital signature schemes. **Post-Quantum Cryptography**, Springer, p. 35–93, 2009.

BUTIN, D.; HÜLSING, A.; MOHAISEN, A.; GAZDAG, S.-L. **Xmss: extended hash-based signatures**. [*S. l.*], 2017.

DANG, Q. **Recommendation for applications using approved hash algorithms**. [*S. l.*]: US Department of Commerce, National Institute of Standards and Technology, 2008.

FERGUSON, N.; SCHNEIER, B.; KOHNO, T. **Cryptography engineering: design principles and practical applications**. [*S. l.*]: John Wiley & Sons, 2011.

GROVER, L. K. A fast quantum mechanical algorithm for database search. **arXiv preprint quant-ph/9605043**, 1996.

HÜLSING, A.; BUTIN, D.; GAZDAG, S.-L.; RIJNEVELD, J.; MOHAISEN, A. Xmss: extended merkle signature scheme. In: **RFC 8391**. [*S. l.*]: IRTF, 2018.

KATZ, J.; LINDELL, Y. **Introduction to modern cryptography**. [*S. l.*]: CRC press, 2014.

KIPNIS, A.; PATARIN, J.; GOUBIN, L. Unbalanced oil and vinegar signature schemes. In: SPRINGER. **Eurocrypt**. [*S. l.*], 1999. v. 99, p. 206–222.

KÖLBL, S.; LAURIDSEN, M. M.; MENDEL, F.; RECHBERGER, C. Haraka v2–efficient short-input hashing for post-quantum applications. **IACR Transactions on Symmetric Cryptology**, p. 1–29, 2016.

LAMPORT, L. **Constructing digital signatures from a one-way function**. [*S. l.*], 1979.

MCELIECE, R. J. A public-key cryptosystem based on algebraic. **Coding Thv**, v. 4244, p. 114–116, 1978.

MCGREW, D.; CURCIO, M.; FLUHRER, S. Hash-based signatures (draft-mcgrew-hash-sigs-06). In: **Crypto Forum Research Group**. [*S. l.*: *s. n.*], 2016.

MERKLE, R. Secrecy, authentication, and public key systems. **Ph. D. Thesis, Stanford University**, 1979.

NIST. **Announcing Request for Nominations for Public-Key Post-Quantum Cryptographic Algorithms**. 2016. Accessed: May 21, 2020. Disponível em: https://csrc.nist.gov/news/2016/public-key-post-quantum-cryptographic-algorithms.

PAAR, C.; PELZL, J. **Understanding cryptography: a textbook for students and practitioners**. [*S. l.*]: Springer Science & Business Media, 2009.

PATARIN, J. The oil and vinegar signature scheme. In: **Dagstuhl Workshop on Cryptography1997**. [*S. l.*: *s. n.*], 1997.

PUB, F. Secure hash standard (shs). **FIPS PUB 180**, v. 4, 2012.

PUB, N. Draft fips pub 202: Sha-3 standard: Permutation-based hash and extendable-output functions. **Federal Information Processing Standards Publication**, 2014.

RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. **Communications of the ACM**, ACM, v. 21, n. 2, p. 120–126, 1978.

SHOR, P. W. Algorithms for quantum computation: Discrete logarithms and factoring. In: IEEE. **Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on**. [*S. l.*], 1994. p. 124–134.