Implementação em software do Esquema de Assinatura Digital de Merkle e suas variantes

Ana Karina D. S. Oliveira[1], **Julio López**[2]

[1]Faculdade de Computação – Universidade Federal do Mato Grosso do Sul (UFMS)
Caixa Postal 549 – 79.070-900 – Campo Grande – MS – Brasil

[2]Instituto de Computação – Universidade Estadual de Campinas(UNICAMP)
Caixa Postal 6.176 – 13083-852 – Campinas – SP– Brasil

anakarina@facom.ufms.br, jlopez@ic.unicamp.br

Abstract. *We describe an efficient software implementation of the Merkle signature scheme and its variants XMSS, CMSS e GMSS. Our implementation is based on hash functions SHA-2 and SHA-3 with security level* 128 *bits and executed on the Intel Core* $i7$ *processor* 2.2 *GHz. The main contributions of this work are a detailed study of the Merkle signature scheme and an optimized implementation for the Intel platform with instructions* $64/128$ *bits.*

Resumo. *Neste trabalho é descrita uma implementação eficiente em software do esquema de assinatura digital Merkle e suas variantes XMSS, CMSS e GMSS. Nossa implementação é baseada nas funções de resumo SHA-2 e SHA-3 com nível de segurança de* 128 *bits e executado em um processador Intel Core* $i7$ 2.2 *GHz. As principais contribuições deste trabalho são um estudo aprofundado do esquema de assinatura digital Merkle e uma implementação otimizada para a plataforma Intel com instruções de* $64/128$ *bits.*

palavras-chave:criptografia pós-quântica, assinaturas digitais, esquema de Merkle.

1. Introdução

Assinatura digital é um método criptográfico de autenticação de informação digital, cujo conceito foi proposto por Whitfield Diffie and Martin Hellman [Diffie and Hellman 1976]. Atualmente, são utilizados esquemas de assinatura digital, tais como RSA [Rivest et al. 1977], *DSA* [NIST 1994] e *ECDSA* [Johnson et al. 2001], que têm sua segurança baseada na dificuldade de fatorar grandes números inteiros e calcular logaritmos discretos. Peter Shor [Shor 1994] introduziu um algoritmo para fatorar inteiros e calcular logaritmos discretos nos grupos relevantes em tempo polinomial, utilizando operações simples em um computador quântico. Dessa forma, a busca de esquemas de assinatura digital resistentes a esses computadores tornou-se relevante.

O esquema de assinatura digital proposto por Merkle *(MSS)* [Merkle 1979] é considerado prático e resistente aos computadores clássicos e quânticos, pois não se conhece, na literatura, uma maneira de aplicar o algoritmo de Shor neste esquema. O MSS utiliza uma função de resumo e sua segurança é baseada na resistência à colisão da função de resumo utilizada. Considerando que uma função de resumo produz strings de tamanho l bits, o número de resumos diferentes que esta função retorna é 2^l. Através

do ataque de aniversário, a probabilidade de encontrar uma colisão é da ordem de $2^{l/2}$ [Katz and Lindell 2008].

Neste trabalho nós fazemos uma apresentação do esquema de Merkle e suas variantes *XMSS*, *CMSS* e *GMSS*. Implementamos os esquemas descritos e mostramos os tempos obtidos em trabalhos anteriores e os tempos obtidos em nossa implementação. Fazemos uma análise comparativa em razão da função de resumo utilizada, da quantidade de assinaturas possíveis, do custo de armazenamento e dos tempos de processamento destes esquemas. Nossa implementação em software utiliza os algoritmos de resumo criptográficos *SHA-2* [NIST 2008] e o novo padrão *SHA-3* [Bertoni et al. 2012], em um computador Intel Core com instruções de 64 e/ou 128 bits. Os conceitos básicos do esquema de Merkle são introduzidos na Seção 2; na Seção 3 descrevemos os esquemas *XMSS*, *CMSS* e *GMSS*. Na Seção 4 especificamos os detalhes de nossa implementação. Apresentamos os resultados obtidos neste trabalho na Seção 5 e a conclusão na Seção 6.

2. Conceitos Básicos do Esquema de assinatura digital de Merkle - MSS

O esquema de assinatura digital *MSS* é baseado em uma árvore binária completa de resumo e em assinaturas *one-time*. A árvore de resumo reduz a validade de uma grande quantidade de chaves de verificação *one-time* para uma única chave, a raiz da árvore.

2.1. Funções de Resumo

O esquema de assinatura de Merkle tem sua segurança baseada na segurança das funções de resumo. Uma função de resumo g tem como entrada uma longa cadeia de bits arbitrária e produz um resultado de tamanho fixo de n bits. Algumas propriedades de funções de resumo são: resistência a preimagem, resistência a segunda preimagem e resistência a colisão. Neste trabalho consideramos as funções *SHA-2* com resumo de 256 bits e a nova função *SHA-3* com resumo de 256 bits.

2.2. Assinatura one-time

Esquemas de assinatura *one-time* apareceram no trabalho de Merkle [Merkle 1979]. Estes esquemas são baseados em funções de resumo e permitem a assinatura de uma única mensagem com o mesmo par de chaves (pública e privada). Merkle utilizou o esquema de assinatura de Winternitz *W-OTS* [Merkle 1987]. Em 2009, foi proposto um esquema de assinatura de Merkle utilizando assinatura *one-time* de Lamport, que produz chaves maiores mas diminui o tempo de assinatura e verificação [Vuillaume et al. 2009].

O esquema *W-OTS* usa uma função de resumo $g : \{0,1\}^* \rightarrow \{0,1\}^n$, que é aplicada várias vezes na chave de assinatura para assinar a mensagem. O parâmetro w denota o número de bits que são processados simultaneamente. Quanto maior for o w menor será a chave de assinatura e maior será o tempo de assinatura e verificação. Em [Dods et al. 2005] foi feita uma análise comparativa do tempo de execução e tamanho das chaves em relação ao parâmetro w.

Geração do par de chaves: Um parâmetro $w \in \mathbb{N}$ é escolhido. A chave de assinatura privada é $x = (x_0, ..., x_{t-1}) \in_{\mathcal{R}} \{0,1\}^{(n,t)}$ onde os x_i são escolhidos aleatoriamente. O tamanho t é obtido calculando $t := t_1 + t_2$, onde $t_1 := \lceil n/w \rceil$ e $t_2 := \lceil (\lfloor log_2 t_1 \rfloor + 1 + w)/w \rceil$. A chave pública de verificação $y = (y_0, ..., y_{t-1})$ é gerada aplicando a função g a cada elemento da chave de assinatura $2^w - 1$ vezes, ou seja $y_i := g^{2^w-1}(x_i)$ para $i := 0, ..., t-1$.

Geração de assinatura: Para gerar uma assinatura do resumo $d = (d_0, ..., d_{n-1})$ da mensagem M, primeiro, adicionamos zeros a esquerda de d, tal que d seja divisível por w, se necessário. Então d é dividido em t_1 blocos de string de bits de tamanho w, resultando em $b = (b_0||...||b_{t_1-1})$, onde $||$ representa a concatenação. As próximas strings de bits b_i são identificadas com inteiros em $\{0, 1, ..., 2^w - 1\}$ e o checksum $c := \sum_{i=t-t_1}^{t-1}(2^w - b_i)$ é calculado. Dividimos a representação binária de c que é menor que $(\lfloor log_2 t_1 \rfloor + 1 + w)$ para os próximos blocos de b. Se for necessário, adicionam-se zeros à esquerda do primeiro bloco, de tal forma que a string seja divisível por w. A string será composta de t_2 blocos $b_{t_1}, ..., b_{t-1}$ de tamanho w. Assim, se calcula a assinatura $sig_i = (g^{b_0}(x_0), g^{b_1}(x_1), ..., g^{b_{t-1}}(x_{t-1}))$.

Verificação de assinatura: Para verificar a assinatura $sig = (sig_0, ..., sig_{t-1})$ da mensagem M, primeiro calculamos os parâmetros $b_0, ..., b_{t-1}$ da mesma forma que durante a geração de assinatura. Para $i := 0, ..., t-1$ calcula-se $sig'_i = g^{2^w-1-b_i}(sig_i)$. Então, se $Y' = (sig'_0||...||sig'_{t-1})$ é igual a $Y = (y_0||...||y_{t-1})$ a assinatura é válida, caso contrário é recusada.

2.3. MSS-Esquema de assinatura digital de Merkle

No esquema de assinatura digital de Merkle descrito a seguir, a chave de assinatura e a chave de verificação *one-time* são as folhas da árvore e, a chave pública, é a raiz. Uma árvore com altura H e 2^H folhas terá 2^H pares de chaves *one-time* públicas e privadas.

2.3.1. Geração de chaves da árvore de Merkle

Para gerar a chave pública *pub*, que é a raiz da árvore de Merkle, primeiro é preciso gerar o par de chaves, pública e privada, *one-time* para cada folha da árvore de Merkle.

Geração do par de chaves: Um algoritmo de assinatura *one-time* gera as chaves públicas X_i e privadas Y_i, para $i := 1, ..., 2^H$.

Geração da chave pública pub O Algoritmo 1 demonstra os passos para gerar a chave pública *pub*. Os parâmetros de entrada são a folha inicial e a altura da árvore.

Algoritmo1: ArvoreDeResumo(folhaIni, alturaMax) [Merkle 1979]

---

1. cria uma pilha $Prox$.
2. **para**($f := folhaIni$, $f < 2^{alturaMax}$, f++) **faça**
   2.1 $no_f := g(Y_f)$: função de resumo g que calcula uma nova folha"
   2.2 empilha no_f na pilha $Prox$
   2.3 **enquanto** os nós do topo da pilha $Prox$ têm alturas iguais **faça**:
   2.3.1 desempilha o valor do nó no_{dir} de $Prox$
   2.3.2 desempilha o valor do nó no_{esq} de $Prox$
   2.3.3 calcula $no_{pai} := g(no_{esq}||no_{dir})$
   2.3.4 **se** altura de $no_{pai} = alturaMax$ **então retorne**(no_{pai})
   senão empilha no_{pai} na pilha $Prox$

Cada nó folha (no_f) recebe o valor de resumo da chave de verificação $g(Y_f)$ daquela folha. Cada nó intermediário (no_{pai}) é o valor resumo da concatenação de seus

dois filhos, esquerdo(no_{esq}) e direito(no_{dir}). Cada vez que uma folha f é calculada e empilhada em $Prox$, o algoritmo verifica se os nós do topo da pilha tem alturas iguais. Se as alturas são iguais, os dois nós são desempilhados, concatenados em no_{pai} e gerado um novo resumo, que será empilhado em $Prox$. O algoritmo termina quando a raiz da árvore é encontrada.

2.3.2. Geração de assinatura:

A mensagem M é assinada com o esquema de assinatura *one-time*, resultando em uma assinatura sig', no qual utiliza a chave X_i para assinar a mensagem M_i [Bernstein et al. 2009]. Uma árvore de autenticação $Atual_i$ é utilizada para guardar os nós no caminho necessários para autenticar uma folha Y_i, reduzindo a necessidade de enviar toda a árvore para o receptor. A assinatura de Merkle é composta de sig' e da correspondente chave de verificação Y_i. Também é preciso incluir o índice i (índice da folha) e o respectivo caminho de autenticação, $Atual = (Atual_0, .., Atual_{H-1})$. Assim, a assinatura $Sig = (i, sig', Y_i, (Atual_0, ..., Atual_{H-1}))$.

O algoritmo clássico do caminho de autenticação

O algoritmo clássico do caminho de autenticação *(Classic Merkle Tree Traversal)* [Merkle 1987] utiliza duas variáveis do tipo pilha: $Atual$ e $Prox$. A pilha $Atual_h$ contém o caminho de autenticação atual e a pilha $Prox$ prepara os próximos nós de autenticação. O caminho $Atual$ é composto pelos valores dos nós $Atual_h$ em cada altura que são os irmãos direitos dos nós no caminho de autenticação que ligam a folha até a raiz da árvore de Merkle. Uma proposta para fazer isso de forma eficiente é guardar o primeiro caminho de autenticação $Atual$ ao executar o Algoritmo 1.

A execução do Algoritmo 1 e do primeiro caminho de autenticação pode ser observado na Figura 1. A Arvore de Resumo mostra a ordem em que os nós são empilhados na árvore. Os nós em cinza mostram o primeiro caminho de autenticação salvo na inicialização.

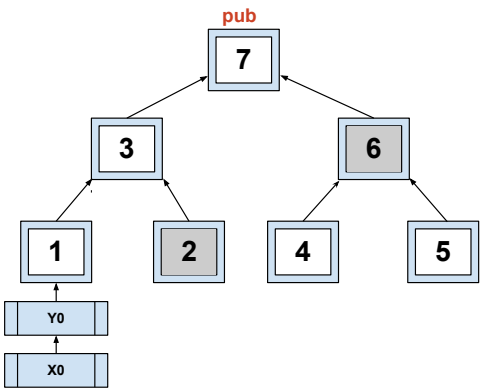


Figura 1. Execução do Algoritmo 1 com o primeiro caminho de autenticação

.

Fase de saída e atualização: O Algoritmo 2 mostra os passos para produzir o próximo caminho de autenticação para a próxima folha f da árvore. A folha é iniciada

com $f := 0$ e a cada assinatura realizada, o Algoritmo 2 é executado. A folha é atualizada em uma unidade e o próximo caminho de autenticação é preparado de forma eficiente, pois somente os nós do caminho que mudam serão atualizados.

Algoritmo2: Percurso(H) [Merkle 1979]

---

1. **para**($f := 0$, $f < 2^H$, f++) faça:
   1.1 **para** ($h := 0, h < H, h++$) **faça**:
   1.1.1**retorne** $Atual_h$ (gera a saída Atual para a assinatura da folha f)
   1.2 **para** ($h := 0, h < H, h++$) **faça**:
   1.2.1 **se** $(f+1)/(2^h) = 0$ **então**:
   1.2.1.1 $Atual_h := Prox_h$ (atualiza $Atual_h$ para as próximas assinaturas)
   1.2.1.2 $no_{Ini} := (f + 1 + 2^h) \oplus 2^h$.
   1.2.1.3 $Prox_h.inicializa(no_{Ini}, h)$.
   1.2.1.4 $Prox_h.atualiza(2)$. (cada Prox recebe duas atualizações)

O Algoritmo 2 atualiza os nós de autenticação através da execução da função $Prox_h.atualiza(2)$, que chama o Algoritmo 1 com o nó e altura selecionada pela função $Prox_h.inicializa(no_{Ini}, h)$. Depois de 2^h rodadas o valor do nó selecionado será completado.

2.3.3. Verificação de assinatura:

De acordo com o método em [Bernstein et al. 2009], o processo de verificação de assinatura consiste de duas etapas: na primeira etapa, a assinatura sig' é verificada utilizando-se a chave de verificação *one-time* Y_i e o respectivo algoritmo *one-time*; na segunda etapa, é preciso validar a chave pública da árvore de Merkle, então o receptor calcula o respectivo caminho de autenticação, construindo o caminho $(p_0, ..., p_H)$ da folha i até a raiz, para toda altura h. O índice i é utilizado para decidir em qual ordem o caminho de autenticação será reconstruído. Inicialmente $p_0 := g(Y_i)$. Para $h := 0, ..., H-1$ executa-se a condição a seguir para calcular o valor resumo da sequência dos nós em cada altura:

se $\lfloor i/(2^{h-1}) \rfloor \equiv 1 \bmod 2$ **então** $p_h := g(Atual_{h-1}||p_{h-1})$

senão se $\lfloor i/(2^{h-1}) \rfloor \equiv 0 \bmod 2$ **então** $p_h := g(p_{h-1}||Atual_{h-1})$.

Ao final, compara-se o valor de p_H com a chave pública conhecida pub. Se o valor for igual, a autenticação é confirmada.

3. Variantes do Esquema de Assinatura de Merkle

3.1. XMSS-eXtended Merkle Signature Scheme

O esquema de assinatura *XMSS* [Buchmann et al. 2011b] é uma modificação do esquema *MSS*, sendo o primeiro esquema de assinatura prático comprovadamente *foward secure* com exigências mínimas de segurança. Este esquema utiliza uma família de funções F e uma família de funções de resumo G. O esquema *XMSS* é eficiente, se G e F são eficientes. Este esquema, é infalsificável sob ataques adaptativos de mensagem escolhida no modelo padrão se G é resistente a segunda preimage e F é pseudo aleatório. Os

parâmetros de *XMSS* são: o parâmetro de segurança $n \in \mathbb{N}$, o parâmetro de Winternitz $w \in \mathbb{N}(w > 1)$, o tamanho da mensagem em bits $m \in \mathbb{N}$, uma família de funções $F_n = \{f_K : \{0,1\}^n \rightarrow \{0,1\}^n | K \in \{0,1\}^n\}$, a altura da árvore $H \in \mathbb{N}$, uma função de resumo g_k escolhida aleatoriamente com distribuição uniforme da família de funções $G_n = \{g_K : \{0,1\}^{2n} \rightarrow \{0,1\}^n | K \in \{0,1\}^n\}$ e a chave de assinatura one-time $x \in \{0,1\}^n$ escolhida aleatoriamente com distribuição uniforme.

A chave de assinatura *one-time* x é utilizada para construir a chave de verificação *one-time* y, através da aplicação da família de funções F_n. Neste trabalho utilizou-se a função $f_K(x) = g(Pad(K)||Pad(x))$, para uma chave $K \in \{0,1\}^n$, $x \in \{0,1\}^n$ e $Pad(z) = (z||10^{b-|z|-1})$ para $|z| < b$, sendo b o tamanho do bloco da função de resumo.

O esquema *XMSS* utiliza uma versão um pouco modificada do esquema de Winternitz [Buchmann et al. 2011a]. Esta modificação permite eliminar a necessidade de uma família de funções de resumo resistente a colisão. Utiliza um caminho através da família de funções em vez de uma avaliação iterada da função de resumo. Para $K, x \in \{0,1\}^n$, $e \in \mathbb{N}$, e $f_K \in F_n$ definiu-se f_K^e como: $K = f_K^0(x)$ e para $e > 0$ definiu-se $K' = f_K^{e-1}(x)$ e $f_K^e(x) = f_{K'}(x)$. Para o parâmetro w de Winternitz, calcula-se: $l_1 = \lceil m/log(w) \rceil$, $l_2 = \lfloor (log(l_1(w-1)))/log(w) \rfloor$+1, $l = l_1 + l_2$.

O esquema *W-OTS* assina mensagens binárias de comprimento m. Os bits da mensagem são processados na base w. A mensagem é da forma M = (M_1 ... M_{l_1}), $M_i \in \{0, ..., w-1\}$. A soma de verificação é $C = \sum_{i=1}^{l_1}(w - 1 - M_i)$ em base de representação w é anexada a M. O tamanho da assinatura, das chaves de assinatura e verificação é de l strings de n bits . A soma de verificação é de comprimento l_2. O resultado é $(b_1, ..., b_l)$. A assinatura de M é: $\sigma = (\sigma_1, ..., \sigma_l) = (f_{sk1}^{b1}(x), ..., f_{skl}^{bl}(x))$.

A árvore *XMSS* é uma modificação da árvore de resumo de Merkle. A árvore de altura H, tem $H+1$ níveis. Utiliza função de resumo g_K e *bitmasks* $(b_{i,j}||b_{r,j}) \in \{0,1\}^{2n}$, escolhidos aleatoriamente. Os nós no nível j, $0 \leq j \leq H$, são denotados por $NODE_{i,j}$, $0 \leq i < 2^{H-j}$, e $0 < j \leq H$. Os nós são calculados como:

$$NODE_{i,j} = g_K((NODE_{2i,j-1} \oplus b_{l,j})||(NODE_{2i+1,j-1} \oplus b_{r,j}))$$

Os bitmasks são a principal diferença das outras árvores de Merkle e permitem substituir a resistência a colisão família função de resumo.

3.2. CMSS-An Improved Merkle Signature Scheme

O esquema *CMSS* [Buchmann et al. 2006] é uma variante do esquema *MSS* que permite aumentar a quantidade de assinaturas de 2^{20} para 2^{40}. Neste trabalho os autores demonstram que *CMSS* é competitivo na prática, apresentando uma aplicação altamente eficiente dentro do *Java Cryptographic Service Provider FlexiProvider* e mostram que a aplicação pode ser usada para assinar mensagens no Microsoft Outlook.

No esquema *CMSS* são construídas duas árvores, uma subárvore e uma árvore principal, cada uma com 2^h folhas, para $h = H/2$. Assim, aumenta o número de assinaturas em relação ao *MSS*, pois o *MSS* se torna impraticável para uma altura $H > 25$, porque aumenta muito a quantidade de chaves privadas a serem armazenadas e, a geração de par de chaves leva muito tempo. Para gerar 2^{20} chaves de assinatura, duas árvores com altura 2^{10} são geradas no *CMSS*, enquanto que no *MSS* uma única árvore com 2^{20} nós é construída. Desta forma, o tempo de geração de chaves é reduzido.

Para melhorar o tempo de geração de assinatura, os autores utilizam o algoritmo de Szydlo [Szydlo 2003] que é mais eficiente. Este algoritmo foi implementado no artigo [Buchmann et al. 2008], no qual a proposta é equilibrar a quantidade de cálculos de folhas em cada caminho de autenticação. Uma pilha mantém nós prontos que serão utilizados em breve e outra pilha armazena os próximos nós que são precalculados.

Para reduzir o tamanho da chave privada, utiliza-se um gerador de números *PRNG* [NIST 2007] no qual somente a semente para o *PRNG* é armazenada. Utilizando uma função de resumo de n bits e o parâmetro de Winternitz t, a chave de assinatura teria $(t * n)$ bits. Com o gerador *PRNG*, é possível guardar somente a semente com n bits.

O esquema *CMSS* usa duas árvores de autenticação *MSS*, uma árvore principal e uma subárvore, A chave pública *CMSS* é a raiz da árvore principal. Os dados são assinados com as folhas da subárvore. Após as 2^h primeiras assinaturas serem geradas, uma nova subárvore é construída e utilizada para gerar as próximas 2^h assinaturas.

Geração de Chaves *CMSS*: Para gerar o par de chaves, o algoritmo de geração de chaves do *MSS* é chamado duas vezes. Inicialmente, a primeira subárvore e seu primeiro caminho de autenticação são gerados. Então, a árvore principal e seu primeiro caminho de autenticação são computados. A chave pública *CMSS* é a raiz da árvore principal. *CMSS* usa o esquema de assinatura *one-time* de Winternitz.

Geração de Assinatura *CMSS*: A geração de uma assinatura *CMSS* é realizada em quatro partes. Primeiro, a assinatura *MSS* do documento d é calculada usando a subárvore. Então, a assinatura *MSS* da raiz da subárvore é calculada usando uma folha árvore principal. Então, a próxima subárvore é parcialmente construída. Finalmente, a chave privada *CMSS* é atualizada para a próxima assinatura. Cada vez que uma nova assinatura *CMSS* é computada, a assinatura da raiz da subárvore atual é recalculada. O tempo necessário para recalcular esta assinatura *MSS* é tolerável.

***CMSS* Verificação de Assinatura**: A verificação de assinatura *CMSS* é realizada em dois passos. Primeiro, os dois caminhos de autenticação são validados, então a validade das duas assinaturas *one-time* são verificadas.

3.3. GMSS-Merkle signatures with virtually unlimited signature capacity

O esquema *GMSS* foi publicado em 2007 [Buchmann et al. 2007], no qual os autores propõem uma alteração no esquema de assinatura Merkle que permite assinar um número ilimitado de mensagens 2^{80} usando um único par de chaves. O esquema *GMSS* foi utilizado para projetar um protocolo cliente servidor para servidores web usando *SSL/TLS* que minimiza a latência e aumenta a resistência a ataques de negação de serviço.

A construção básica de *GMSS* consiste de uma árvore com T camadas. Subárvores em camadas diferentes podem ter alturas diferentes. Para reduzir o custo de geração de uma assinatura, o esquema *GMSS* usa a geração de uma assinatura de forma distribuída, distribuindo a geração de assinatura em cada subárvore em várias assinaturas. Este esquema também possibilita escolher parâmetros w de Winternitz diferentes para subárvores em camadas diferentes, produzindo assim, assinaturas menores.

Geração de chaves GMSS: Para cada subárvore, o algoritmo de geração de chaves, *W-OTS*, gera as chaves de assinatura e o Algoritmo 1 da Árvore de Resumo calcula as raízes das árvore $Root_{\tau,0}$. Os primeiros caminhos de autenticação de cada subárvore

são salvos durante na geração de $Root$. Depois as assinaturas Sig_τ das T árvores Merkle que serão usadas para a primeira assinatura são calculadas. Como os valores mudam com menos frequência para as camadas superiores, a precomputação pode ser distribuída por várias etapas, resultando em uma melhora significativa da velocidade de assinatura e permitindo escolher grandes parâmetros w de Winternitz, que resulta em assinaturas menores. Para garantir tamanhos pequenos de chaves privadas, foi utilizado *PRNG*, onde somente a semente do *PRNG* precisa ser armazenada.

Geração de assinatura GMSS: A raiz de uma árvore filha é assinada com a chave de assinatura *one-time* correspondente a uma certa folha de sua árvore mãe. $Root_\tau$ denota a raiz da árvore τ. Sig_τ denota a assinatura *one-time* de $Root_\tau$, que é gerado usando a folha l do pai de τ. Os resumos da mensagem d são assinados utilizando as folhas das árvores Merkle sobre a camada T mais profunda. O número de mensagens que podem ser assinadas com um par de chaves *GMSS* é $S = 2^{h_1+...+h_T}$. A assinatura *GMSS* consiste do índice s, das assinaturas *one-time* Sig_d e $Sig_{\tau_{i,j_i}}$ para $i = 2, ..., T$ e dos caminhos de autenticação $Atual_{\tau_{i,j_i},l_i}$ para $i = 1, ..., T$.

Na geração de assinatura também são calculados as raízes $Root_{\tau_{i,1}}$ e caminhos de autenticação $Atual_{\tau_{i,1},0}$, das árvores seguintes $\tau_{i,1}$, onde $i = 2, ..., T$. A geração de assinatura pode ser dividida em duas partes. A primeira parte, parte *online*, calcula Sig_d e a assinatura. A segunda parte, parte *ofline*, precalcula os caminhos de autenticação e assinaturas *one-time* das raízes necessárias para as próximas assinaturas.

Verificação de assinatura GMSS: O processo de verificação de *GMSS* é essencialmente o mesmo que para *MSS* e *CMSS*. Primeiro, verifica-se a assinatura de uma só vez Sig_d do digest d usando o esquema *one-time* de Winternitz. Então, o verificador valida as chaves públicas das raízes das subárvores e da árvore principal.

4. Especificação e Implementação

Implementamos os algoritmos descritos anteriormente: *MSS*, *XMSS*, *CMSS* e *GMSS*. Como o esquema *GMSS* permite a criação de 2 ou 4 subárvores, denotamos *GMSS* T=2 (w_2, w_1) para 2 subárvores e *GMSS T=4* (w_4, w_3, w_2, w_1) para 4 subárvores, sendo w_1 a árvore principal (pai) da camada mais acima e w_2,w_3 e w_4 as subárvores (filhas) das camadas mais abaixo. Nas implementações *MSS*, *CMSS* e *GMSS* usamos o esquema *W-OTS* [Merkle 1979], que foi descrito na Seção 2.2. Para o esquema *XMSS*, usamos uma variante do esquema *W-OTS* proposto em [Buchmann et al. 2011a], descrito na Seção 3.1. Os parâmetros w de Winternitz escolhidos foram de 3 a 9. Se $w_2 = w_1$, então *GMSS T=2* é igual ao esquema *CMSS*. Para gerar as chaves de assinatura, utilizamos o *PRNG* conforme [NIST 2007], permitindo guardar chaves de assinatura menores.

Em todas as implementações, utilizamos o algoritmo proposto em [Buchmann et al. 2008] que calcula os próximos caminhos de autenticação de forma eficiente. Conforme proposto no esquema *GMSS* [Buchmann et al. 2007], dividimos a geração de assinatura em duas partes: A primeira parte, assinatura *online*, calcula a assinatura *W-OTS*. A segunda parte, assinatura *offline*, precalcula os caminhos de autenticações para a próxima assinatura. O primeiro caminho de autenticação foi calculado no momento da geração de chaves.

Foram utilizadas árvores de altura H igual a 20, 40 e 80 que geram uma quantidade

de 2^H assinaturas. Na Tabela 1 indicamos que algoritmos foram implementados para cada altura da árvore.

	H=20	H=40	H=80
MSS	implementado		
XMSS	implementado		
CMSS	implementado	implementado	
GMSS T=2	implementado	implementado	
GMSS T= 4	implementado	implementado	implementado

Tabela 1. Implementações realizadas

Para a função de resumo *SHA-2(256)* utilizamos o código otimizado em C de Kevin Springle et al., disponível em [Springle et al. 2012]. Também utilizamos o código *SHA-2(256)* 4 SMS (128 bits) em C implementado por Jeremi Gosney, disponível em http://www.openwall.com. Para a função de resumo *SHA-3(256)* 64 e 128 bits, utilizamos o código otimizado em C implementado por Guido Bertoni et al., disponível em http://bench.cr.yp.to/supercop.html.

Em razão da utilização de blocos de tamanho fixo, como no caso de cada elemento da chave de assinatura que tem exatamente 256 bits, implementamos uma função de pré-processamento reduzida no *SHA-2* para fazer o PAD da mensagem. Os bits de controle ficam sempre na mesma posição, e a implementação foi efetuada sem loops ou condições, o que permitiu ganho no tempo de execução.

Em alguns pontos da implementação, utilizamos o *OpenMP* para paralelizar as tarefas e diminuir o tempo de execução. Por exemplo, o esquema de assinatura *W-OTS* executa $2^w - 1$ vezes a função de resumo em cada elemento x_i da chave de assinatura X, onde $i = 0, ..., t - 1$, criamos 3 threads para executar os t valores da chave de assinatura em paralelo, como mostrado no Algoritmo 3:

Algoritmo3: Paralelização da Geração da Chave de Verificação

1. #pragma omp parallel for num_threads(3)
2. **para**$(i = 0; i < t; i++)$ **faça**
   - 1.1 $Y[i] = X[i]$;
   - 1.2 **para**$(k[i] = 0; k[i] < (2^w - 1); k[i]++)$ **faça**
     - 1.2.1 $Y[i] = g(Y[i])$;

Para a geração de assinatura e verificação de assinatura, utilizamos o mesmo procedimento do Algoritmo 3. A diferença está no segundo laço, no qual o número de aplicações da função de resumo depende dos bits da mensagem. Para os algoritmos que criam subárvores *CMSS* e *GMSS*, também criamos threads para distribuir a execução de cada subárvore para um *thread*, executando assim os cálculos de cada subárvore ao mesmo tempo.

Os experimentos deste trabalho foram feitos usando uma máquina Intel Core $i7 - 2670$ QMCPU, 2.20 GHz com 6 GB de RAM, 500 GB de disco com o turbo boots desligado. O sistema operacional utilizado é o Linux Ubuntu 12.04. A linguagem utilizada foi o *C* e o compilador *gcc* $4.6.3$.

5. Resultados experimentais

Nesta seção, comparamos os resultados da nossa implementação em software dos esquemas *MSS*, *CMSS*, *GMSS* e *XMSS*. Mostramos os resultados obtidos em alguns trabalhos anteriores e comparamos com os nossos resultados. Comparamos os tamanhos das chaves pública, privada e de assinatura de cada esquema em relação ao parâmetro w, altura H e função de resumo escolhidos. Comparamos também os tempos necessários para geração de um par de chaves, geração de assinatura e verificação de assinatura de cada esquema implementado com as funções de resumo *SHA-2* e *SHA-3*.

5.1. Trabalhos Anteriores

Na Tabela 2 mostramos os resultados de tempos obtidos em trabalhos publicados anteriormente: *CMSS*, *GMSS*, *XMSS*, *HTG* e *MCP*. Definimos: (t_{chaves}) tempo para gerar as chaves de assinatura, verificação e pública. O tempo para gerar e verificar assinatura são respectivamente: (t_{sig}) e (t_{ver}).

Implementações em Software: O esquema *CMSS* [Buchmann et al. 2006] foi testado em um computador Pentium M 1.73GHz, 1GB de RAM rodando no Microsoft Windows XP, o esquema *GMSS* [Buchmann et al. 2007] em um computador Pentium dual-core 1.8 GHz e o esquema *XMSS* [Buchmann et al. 2011b] em um computador Intel(R) Core(TM) $i5$ $M540$, 2.53GHz com tecnologia Infineon.

Implementações em Hardware(arquitetura Novel em uma plataforma FPGA Virtex-5): o trabalho *HTG(Hash Tree Generator)* [Shoufan and Huber 2010] apresenta um módulo para acelerar a geração de chaves da árvore de Merkle *MSS* com 8 instâncias do módulo *Feedback Hash Module*; e o trabalho *MCP(Merkle Cryptoprocessor)* [Shoufan et al. 2011], apresenta um criptoprocessador para o esquema *CMSS*.

Esquema	**hash**	**H**	**w**	**t**$_{chaves}$	**t**$_{sig}$	**t**$_{ver}$
CMSS	SHA2	20	(3,3)	7.5 s	23.3 ms	3.7 ms
CMSS	SHA2	20	(4,4)	10.2 s	31.6 ms	5.1 ms
CMSS	SHA2	40	(3,3)	120.7 min	40.9 ms	3.7 ms
CMSS	SHA2	40	(4,4)	165.6 min	55.8 ms	5.1 ms
GMSS T=2	SHA1	40	(9,3)	390 min	10.7 ms	10.7 ms
GMSS T=4	SHA1	80	(7,7,7,3)	592 min	10.1 ms	10.1 ms
XMSS	SHA2	20	4	408.6 s	6.3 ms	0.51 ms
XMSS	SHA2	20	16	466.2 s	7.0 ms	0.52 ms
HTG	SHA2	20	4	146 s	-	-
HTG	SHA2	20	6	403 s	-	-
MCP	SHA2	30	4	820 ms	2.7 ms	1.7 ms
MCP	SHA2	30	6	2380 ms	7.4 ms	4.7 ms

Tabela 2. Comparação de tempos de trabalhos anteriores

5.2. Nossa implementação

A seguir mostramos os resultados de nossa implementação. A Tabela 3 mostra os tamanhos das chaves e tempos de execução dos algoritmos implementados com a função de resumo *SHA-2*. Definimos os custos de armazenamento como: (C_{pub}) tamanho da chave pública, (C_{priv}) tamanho da chave privada, e (C_{sig}) tamanho da chave de assinatura. Como

Esquema	**H**	**w**	C_{pub}	C_{priv}	C_{sig}	t_{chaves}	t_{sigOnl}	$t_{sigOffl}$	t_{ver}
MSS	20	3	32	1316	3556	243.5 s	0.13 ms	0.01 ms	0.11 ms
MSS	20	4	32	1316	2820	311.5 s	0.17 ms	0.01 ms	0.16 ms
CMSS	20	(3,3)	32	2056	6472	0.30 s	0.13 ms	0.25 ms	0.28 ms
CMSS	20	(4,4)	32	2056	5000	0.37 s	0.17 ms	0.32 ms	0.40 ms
CMSS	40	(3,3)	32	3976	7112	269 s	0.14 ms	0.26 ms	0.30 ms
CMSS	40	(4,4)	32	3976	5640	384 s	0.17 ms	0.32 ms	0.40 ms
GMSS T=2	40	(9,3)	32	3976	5224	48.1 min	0.16 ms	0.27 ms	4.98 ms
GMSS T=4	80	(3,3,3,3)	32	9232	14224	570 s	0.14 ms	0.29 ms	0.45 ms
GMSS T=4	80	(7,7,7,3)	32	9232	10864	50.3 min	0.13 ms	0.25 ms	2.3 ms
XMSS	20	4	1696	1316	4932	293.6 s	0.25 ms	0.01 ms	0.34 ms

Tabela 3. Tamanhos em bytes, tempos em (ms,s,min) e função SHA-2(256)

o tempo de assinatura foi dividido em duas etapas, definidos como: (t_{sigOnl}) tempo de assinatura *Online*, ($t_{sigOffl}$) tempo de assinatura *Offline*.

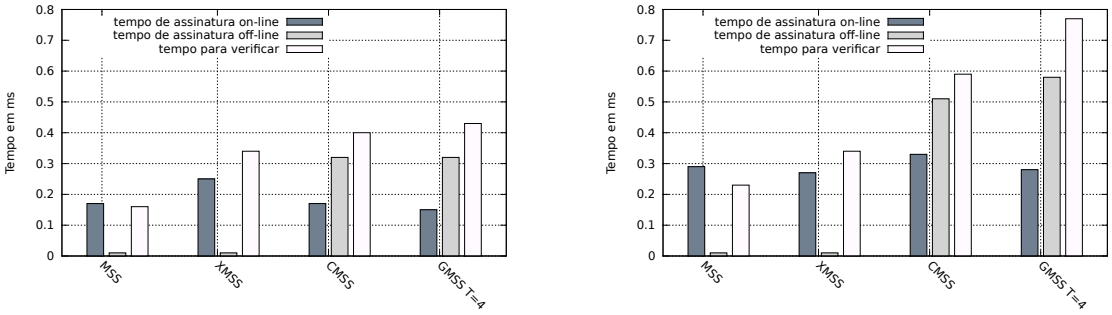
Observamos, pela Tabela 3, que o tamanho das chaves públicas tem 32 bytes, exceto para o esquema *XMSS* que guarda os *bitmasks* das árvores Tree e LTree junto com a chave pública. A chave privada é menor nos esquemas *MSS* e *XMSS*, pois nos outros esquemas é preciso guardar informações de 2 ou mais árvores. A chave de assinatura é menor no esquema *MSS*, pois guarda a assinatura de uma única árvore. Como o esquema *XMSS* utiliza uma versão modificada do esquema de Winternitz, a assinatura neste esquema é maior que no *MSS*.

Os algoritmos *CMSS* e *GMSS* para $H = 20$ tem melhores tempos de geração de chaves em relação aos outros algoritmos, por não precisarem gerar todas as chaves de assinatura inicialmente. Para $H = 20$ e $w = 4$, o esquema *XMSS* teve o pior tempo de geração de chaves (293.6 s) por ter que gerar bits adicionais *(bitmasks)*.

Utilizamos a implementação do *SHA-2(256)* 4 SMS (128 bits) para gerar a chave de verificação de Winternitz no esquema *MSS*. Paralelizamos a aplicação da função de resumo para 4 elementos por vez, em vez de aplicar a função de resumo em um elemento de cada vez. Conseguimos melhorar o tempo de geração de chaves no MSS com altura $H = 20$ e $w = 4$ de (311.5 s) para (195.6 s).

Nos gráficos da Figura 2 apresentamos os tempos de assinatura e verificação dos esquemas com $w = 4$, $H = 20$ utilizando as funções *SHA-2* e *SHA-3* 64 bits. O esquema *MSS* teve os melhores tempos de assinatura e verificação, porque somente um caminho de autenticação precisa ser atualizado a cada assinatura e na verificação uma única árvore precisa ser checada. Porém, o *MSS* é recomendado somente para aplicações que necessitam até 2^{20} chaves de assinatura, se tornando ineficiente para gerar mais chaves por consumir muitos recursos de memória. Mesmo que a geração do par de chaves no *MSS* consuma muito tempo, isto não impede sua utilização, já que a geração do par de chaves tem que ser realizada apenas uma vez. Se forem necessárias mais chaves de assinatura, os algoritmos *CMSS* e *GMSS* são recomendados. Estes algoritmos diminuem o tempo de geração de chaves e aumentam a possibilidade de chaves de assinatura para 2^{40} e 2^{80} respectivamente.

Os testes efetuados com a função de resumo *SHA-3* 64 e 128 bits, mostram um pequeno ganho no tempo de execução para a implementação de 128 bits. Observamos na



(a) SHA-2 (b) SHA-3

Figura 2. Tempos de assinatura e verificação com w=4 e H=20

Tabela 4 o resultado obtido com a implementação do esquema *MSS* para uma árvore de altura $H = 20$ e parâmetro Winternitz $w = 4$.

	t_{chaves}	t_{sigOnl}	$t_{sigOffl}$	t_{ver}
SHA-3 (64 bits)	469.37 s	0.27 ms	0.01 ms	0.21 ms
SHA-3 (128 bits)	449.93 s	0.25 ms	0.01 ms	0.20 ms

Tabela 4. Comparação de tempos no MSS com a função SHA-3(64 e 128 bits)

O gráfico da Figura 3 mostra os resultados com o *GMSS* $T = 2$ e $H = 40$ utilizando parâmetros w diferentes com a função *SHA-2*.

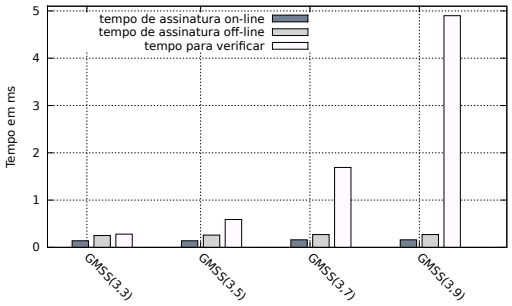
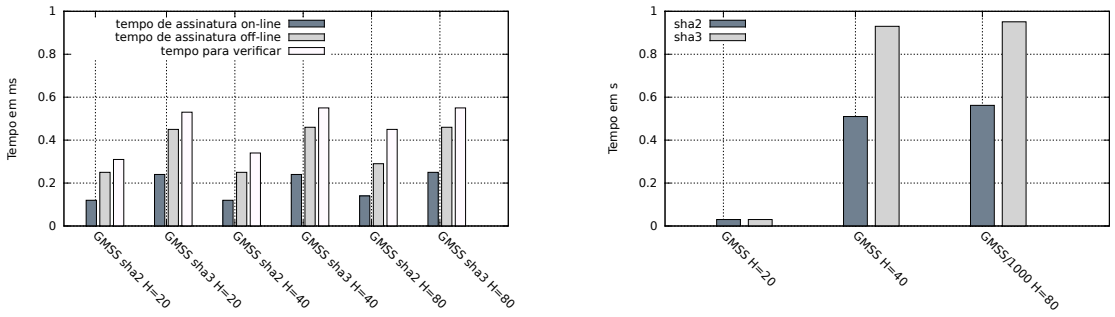


Figura 3. Comparação dos tempos de assinatura e verificação no *GMSS* para diferentes parâmetros w

Conforme aumentamos o parâmetro w, aumentamos os tempos de processamento, porém o tamanho da assinatura fica menor. No *MSS*, apresentado na Tabela 3, observamos que se $w = 3$, o tamanho da assinatura é 26.39% maior que se $w = 4$ e o tempo para assinar é 23.52% menor. No *GMSS*, apresentado na Figura 3, o tempo de verificação aumenta conforme aumentamos o parâmetro w_2, mas o tempo de assinatura se mantém estável, pois o parâmetro w_2 não é utilizado em todas as assinaturas, somente quando uma subárvore é descartada.

Na Figura 4, comparamos os tempos no esquema *GMSS* $T = 4$ e $w = (3, 3, 3, 3)$ para assinar e verificar com as funções de resumo *SHA-2* e *SHA-3* 64 bits para diferentes alturas. Observamos no gráfico (a) que mesmo se aumentamos a altura da árvore, o esquema *GMSS* mantém a sua eficiência na geração de assinatura. A verificação para $H =$

80 foi um pouco pior em relação a alturas menores, por ter que verificar as assinaturas das 4 subárvores e suas raízes. O tempo de geração de chaves neste esquema aumenta quando aumentamos o tamanho da árvore, por ter que gerar mais chaves de assinatura.



(a) Assinatura e Verificação (b) Geração de Chaves

Figura 4. Comparação do GMSS com SHA2/SHA3 para H=20,40,80

6. Conclusão

Mostramos neste trabalho que o esquema de assinatura digital de Merkle e suas variantes são uma alternativa interessante aos tradicionais esquemas como o *RSA* e *ECDSA*, pois os resultados das análises de nossas implementações demonstram que os esquemas *MSS*, *CMSS*, *GMSS* e *XMSS* possuem tamanhos de chaves suportáveis para os computadores atuais. Nossos resultados experimentais mostram que a geração e verificação de assinatura apresentaram tempos razoáveis. No contexto da criptografia pós-quântica, a grande vantagem da utilização destes esquemas é a não vulnerabilidade ao algoritmo de Shor e a possibilidade de resistência ao advento dos computadores quânticos, o que torna viável a sua utilização em aplicações práticas, ultrapassando o plano eminentemente teórico.

A limitação da quantidade de chaves que utilizam a mesma chave pública é a maior barreira para a popularização dos esquemas, mas é possível que, para muitas aplicações, o limite de 2^{20} seja suficiente ou dure um período de tempo grande até que seja substituído por outro esquema ou outra chave pública. Além disso, os limites de 2^{40} e 2^{80} no *GMSS* mostram-se grande o bastante para quase todas as aplicações, sendo o último considerado de tamanho ilimitado.

Referências

Bernstein, D., Buchmann, J., and Dahmen, E. (2009). Post-quantum cryptography. pages 35–93. Springer.

Bertoni, G., J.Daemen, M.Peeters, and Assche1, G. (2012). The keccak sponge function family SHA-3. http://keccak.noekeon.org/.

Buchmann, J., Coronado, C., Dahmen, E., Döring, M., and Klintsevich, E. (2006). CMSS– an improved merkle signature scheme. In *Progress in Cryptology – INDOCRYPT 2006, LNCS 4329*, pages 349–363. Springe-Verlag.

Buchmann, J., Dahmen, E., Ereth, S., Hülsing, A., and Rückert, M. (2011a). On the security of the winternitz one-time signature scheme. In *German Research*, pages 1–17.

Buchmann, J., Dahmen, E., and Hülsing, A. (2011b). XMSS-a practical secure signature scheme based on minimal security assumptions. In *Cryptology ePrint Archive - Report 2011/484*. ePrint.

Buchmann, J., Dahmen, E., Klintsevich, E., Okeya, K., and Vuillaume, C. (2007). Merkle signatures with virtually unlimited signature capacity. In *Applied Cryptography and Network Security - ACNS 2007, LNCS 4521*, pages 31–45. Springer.

Buchmann, J., Dahmen, E., and Schneider, M. (2008). Merkle tree traversal revisited. In *Proceedings of the 2nd International Workshop on Post-Quantum Cryptography*, pages 63–78. Springer-Verlag.

Diffie, W. and Hellman, M. E. (1976). New directions in cryptography. In *IEEE Trans. Information Theory*, pages 644–654. IT-22.

Dods, C., Smart, N., and Stam, M. (2005). Hash-based digital signature schemes. In *In Cryptography and Coding, LNCS 3796*, pages 96–115. Springer.

Johnson, D., Menezes, A., and Vanstone, S. (2001). Elliptic Curve Digital Signature Algorithm ECDSA. page 36–63. Springer-Verlag.

Katz, J. and Lindell, Y. (2008). Introduction to modern cryptography. pages 127–133. Chapman.

Merkle, R. (1987). A digital signature based on a conventional encryption function. In *Proceedings of Crypto '87*, pages 369–378. Springer.

Merkle, R. C. (1979). *Secrecy, Authentication, and Public Key Systems*. Stanford Ph.D. thesis.

NIST (1994). Digital Signatures Algorithm (DSA). FIPS-186, http://www.itl.nist.gov/fipspubs/fip186.htm.

NIST (2007). Digital Signature Standard (DSS). FIPS PUB-186-2, http://csrc.nist.gov/publications/fips.

NIST (2008). Sha-256. FIPS PUB 180-3, http://csrc.nist.gov/publications/fips.

Rivest, R., Shamir, A., and Adleman, L. (1977). A method for obtaining digital signatures and public-key cryptosystems. pages 120–126. Communications of the ACM.

Shor, P. (1994). *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*. IEEE Computer Society Press.

Shoufan, A. and Huber, N. (2010). A fast hash tree generator for merkle signature scheme. In *Circuits and Systems (ISCAS)*, pages 3945–3948. IEEE International Symposium.

Shoufan, A., Huber, N., and Molter, H. (2011). A novel cryptoprocessor architecture for chained merkle signature scheme. In *Microprocessors and Microsystems*, pages 34–47. Elsevier.

Springle, K., Dai, W., Pavlov, I., and Collin, L. (2012). Modified version of the sha-256. http://svn.r-project.org/R/trunk/src/extra/xz/check/sha256.c.

Szydlo, M. (2003). Merkle tree traversal in log space and time. In *Prepint version, 2003*.

Vuillaume, C., Okeya, K., Dahmen, E., and Buchmann, J. (2009). *Public Key Authentication with Memory Tokens*. Development.

