XXXI Congresso de Iniciação Científica Unicamp 2023

Implementação de algoritmo de assinatura digital pósquântica em dispositivos com recursos limitados

Rodrigo Duarte de Meneses **Marco Aurélio Amaral Henriques**

[1]Faculdade de Engenharia Elétrica e de Computação
Universidade Estadual de Campinas (Unicamp) – Campinas, SP – Brasil

r197962@dac.unicamp.br maah@unicamp.br

Palavras-chave: assinatura digital, criptografia pós-quântica, CRYSTALS-Dilithium

1. Introdução

Com os desenvolvimentos recentes da computação quântica e a criação do algoritmo de Shor [Shor 1997], o ataque aos algoritmos de assinatura digital RSA e ECDSA passaram a ser viáveis em tempo polinominal. Assim, a segurança antes atribuída aos criptossistemas mais amplamente utilizados é colocada em risco, comprometendo grande parte das aplicações em segurança de dados. O projeto de esquemas criptográficos resistentes aos ataques de um computador quântico define a área da criptografia pós-quântica.

Em 2022, após seis anos de estudo, o National Institute of Standards and Technology (NIST) padronizou os algoritmos pós-quânticos CRYSTALS-Kyber (Cryptographic Suite for Algebraic Lattices), como algoritmo para encriptação assimétrica, e FALCON, CRYSTALS-Dilithium e SPHINCS+, como algortimos de assinatura digital [NIST 2022]. Mais especificamente, o NIST indica a utilização do Dilithium como algoritmo primário, apontando o FALCON como uma alternativa para aplicações que exijam menores tamanhos de assinatura. O algoritmo SPHINCS+, apesar de padronizado, ficou como uma opção de reserva, visto que demanda muitos recursos computacionais.

Neste trabalho, exploramos estudos que apontam para um overhead computacional associado às funções de hash criptográfico da família SHA-3, amplamente utilizadas no Dilithium [Aumasson 2019]. A substituição das funções SHAKE dessa família pelas versões com redução no número de rodadas TurboSHAKE promete uma aceleração em software do algoritmo sem comprometer a sua segurança, o que facilita uma implementação em sistemas com recursos limitados [Bertoni et al. 2023]. No texto são discutidos os impactos observados em cada função do Dilithium a partir dessa otimização.

2. Assinatura sobre reticulados

2.1. Reticulados e o problema MLWE

O problema matemático sobre o qual o Dilithium adquire sua segurança é o problema Module Learning With Errors (MLWE), definido sobre um tipo de estrutura algébrica abstrata chamada reticulado. Um reticulado é um espaço vetorial discreto, composto por um conjunto S de pontos distribuídos em um espaço n-dimensional. Cada ponto em S pode ser expresso

em termos de uma base de n vetores em S linearmente independentes. Naturalmente, existem diferentes bases possíveis para um dado reticulado, permitindo a definição de bases boas ou ruins – dependendo do quão trivial é expressar os pontos em S em termos de uma combinação linear dos vetores desta base.

A partir dessas propriedades, é possível definir vários problemas sobre essas estruturas algébricas que são de interesse particular para algoritmos de criptografia, como o problema Learning With Errors (LWE). Este problema consiste na resolução de equações matriciais, análogas à combinação linear de vetores em um reticulado S, com a inclusão de um ruído que impossibilita a solução do sistema de forma trivial utilizando o método de eliminação de Gauss-Jordan. Assim, dado $\mathbb{Z}_q$ um anel de inteiros módulo q e $\mathbb{Z}_q^n$ um conjunto de n-vetores sobre $\mathbb{Z}_q$, o problema consiste na dificuldade de se distinguir um vetor qualquer $\mathbf{t} \in \mathbb{Z}_q^n$ de um vetor da forma $\mathbf{t}' = \mathbf{A} \cdot \mathbf{s_1} + \mathbf{s_2}$, onde $\mathbf{s_1}$ e $\mathbf{s_2}$ pertencem a $\mathbb{Z}_q^n$ e $\mathbf{A}$ é uma matriz com entradas em $\mathbb{Z}_q$.

O problema MLWE é uma generalização do LWE em que tomamos um anel polinomial $\mathbb{Z}_q[X]/(X^n + 1)$ ao invés do anel de inteiros módulo q – permitindo uma maior flexibilidade de alteração do nível de segurança sem modificar a aritmética subjacente. Esses problemas são considerados de difícil resolução mesmo para um computador quântico; portanto, sua aplicação em algoritmos pós-quânticos é justificável. No Dilithium, os parâmetros de construção do anel polinomial são fixos, e todas as operações são realizadas sobre o anel $R = \mathbb{Z}_q[X]/(X^{256} + 1)$, com $q = 2^{23} - 2^{13} + 1$.

2.2. CRYSTALS-Dilithium

O CRYSTALS-Dilithium é um algoritmo de assinatura digital com design baseado no esquema "Fiat-Shamir with Aborts", consistindo em três funções principais – geração de chaves (KeyGen), assinatura (Sign) e verificação (Verify), cujos pseudocódigos (sem a compressão da chave pública) estão apresentados nas Figuras 1, 2 e 3, respectivamente.

Gen
01 $\mathbf{A} \leftarrow R_q^{k \times \ell}$
02 $(\mathbf{s}_1, \mathbf{s}_2) \leftarrow S_\eta^\ell \times S_\eta^k$
03 $\mathbf{t} := \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$
04 **return** $(pk = (\mathbf{A}, \mathbf{t}), sk = (\mathbf{A}, \mathbf{t}, \mathbf{s}_1, \mathbf{s}_2))$

Figura 1. Pseudocódigo para função de geração de chaves do Dilithium.

Na função KeyGen, utiliza-se uma semente aleatória ζ de 256 bits para geração dos parâmetros públicos $\mathbf{A}$ e $\mathbf{t}$, e dos parâmetros privados $\mathbf{s_1}$ e $\mathbf{s_2}$. Tanto os vetores $\mathbf{s_1}, \mathbf{s_2}$ e $\mathbf{t}$ quanto a matriz $\mathbf{A}$ têm elementos que são polinômios em R. Em particular, a geração e armazenamento da matriz pública $\mathbf{A}$ é um processo custoso em termos de memória e processamento; por esse motivo, na prática armazenamos somente a semente e geramos a matriz $\mathbf{A}$ conforme a demanda.

Na função Sign, é gerado um vetor de mascaramento $\mathbf{y}$ e computado o vetor $\mathbf{Ay}$, definindo-se $\mathbf{w}_1$ como os "bits de mais alta ordem" de seus coeficientes. Gera-se então o desafio c a partir do hash da mensagem M a ser assinada concatenada à $\mathbf{w}_1$. Em seguida é produzida a potencial assinatura $\mathbf{z}$ utilizando a chave privada sk. Neste momento, caso $\mathbf{z}$ já fosse entregue como saída, existiria a possibilidade de vazamento da chave privada. Por

$\underline{\mathsf{Sign}(sk, M)}$
05 $\mathbf{z} := \perp$
06 **while** $\mathbf{z} = \perp$ do
07 $\quad \mathbf{y} \leftarrow S^{\ell}_{\gamma_1 - 1}$
08 $\quad \mathbf{w}_1 := \mathsf{HighBits}(\mathbf{Ay}, 2\gamma_2)$
09 $\quad c \in B_\tau := \mathrm{H}(M \parallel \mathbf{w}_1)$
10 $\quad \mathbf{z} := \mathbf{y} + c\mathbf{s}_1$
11 $\quad$ **if** $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ or $\|\mathsf{LowBits}(\mathbf{Ay} - c\mathbf{s}_2, 2\gamma_2)\|_\infty \geq \gamma_2 - \beta$, then $\mathbf{z} := \perp$
12 **return** $\sigma = (\mathbf{z}, c)$

Figura 2. Pseudocódigo para função de assinatura digital do Dilithium.

esse motivo, inclui-se uma verificação chamada de *rejection sampling* que garante que a assinatura não revelará informações sobre os parâmetros privados. Por fim, a saída é dada como o conjunto do desafio c e a assinatura $\mathbf{z}$ após o *rejection sampling* não ter detectado nenhum problema de vazamento.

$\underline{\mathsf{Verify}(pk, M, \sigma = (\mathbf{z}, c))}$
13 $\mathbf{w}'_1 := \mathsf{HighBits}(\mathbf{Az} - c\mathbf{t}, 2\gamma_2)$
14 **if return** $[\![\|\mathbf{z}\|_\infty < \gamma_1 - \beta]\!]$ **and** $[\![c = \mathrm{H}(M \parallel \mathbf{w}'_1)]\!]$

Figura 3. Pseudocódigo para função de verificação do Dilithium.

A função Verify calcula $\mathbf{w}'_1$ como os bits de mais alta ordem dos resultado $\mathbf{Az} - c\mathbf{t}$. Deve-se destacar o fato de que para que haja a correta verificação da assinatura, devemos ter $\mathbf{w}'_1 = \mathbf{w}_1$ e portanto HighBits($\mathbf{Az} - c\mathbf{t}$) = HighBits($\mathbf{Ay}$). Importante notar que $\mathbf{Az} - c\mathbf{t} = \mathbf{Ay} - c\mathbf{s}_2$ e que a condição para assinatura válida é $\mathrm{HighBits}(\mathbf{Az} - c\mathbf{t}) = \mathrm{HighBits}(\mathbf{Ay} - c\mathbf{s}_2)$, onde o tamanho reduzido do vetor de ruídos $\mathbf{s_2}$ é essencial para a corretude da verificação.

3. SHAKE e TurboSHAKE

As duas principais operações realizadas pelo Dilithium são expansões de parâmetros aleatórios através de XOFs (eXtendable Output Function), e multiplicações dentro do anel polinomial R, para as quais utiliza-se a NTT (Number Theoretic Transform). Dessa forma, a variação do nível de segurança do Dilithium consiste somente em variar o número de operações realizadas sobre o anel polinomial e as expansões por XOF, o que confere grande flexibilidade de aplicação do algoritmo. O Dilithium usa as funções hash criptográficas da família SHA-3, segundo a norma FIPS 202 do NIST. Em particular, o algoritmo faz uso do SHAKE-128 e SHAKE-256 como XOF para expansão de parâmetros a partir de uma semente de entrada. No entanto, existe um overhead computacional nessas funções hash, de forma que implementações otimizadas passaram a surgir como uma alternativa mais rápida que não comprometa a segurança [Aumasson 2019].

Uma alternativa interessantes é o TurboSHAKE [Bertoni et al. 2023], implementação com rodadas reduzidas (12 rodadas, em comparação às 24 do SHAKE tradicional), adotado no algoritmo KangarooTwelve, que implementa a permutação Keccak-p[1600, 12] com maior velocidade comparada às funções SHA-3 e SHAKE. Utilizando a mesma arquitetura em esponja do SHAKE, o TurboSHAKE possui também versões de 128 e 256 bits, permitindo a

substituição do SHAKE-128 e SHAKE-256 por suas respectivas versões TurboSHAKE com a mesma segurança.

4. Metodologia experimental

Realizamos a medição de ciclos de CPU e uso de RAM para cada uma das três funções: Keygen, Sign e Verify. Comparamos o impacto da substituição das XOFs SHAKE-128 e SHAKE-256 pelas versões otimizadas TurboSHAKE-128 e TurboSHAKE-256, e denotamos por v.0 e v.1 as versões padrão e otimizada do algoritmo, respectivamente. Realizamos 1500 iterações para obtenção de dados, incluindo a geração de chaves, assinatura e verificação para uma mensagem aleatória de 59 bytes. Repetimos todas as medições para os níveis se segurança 2, 3 e 5, conforme definido na documentação oficial do Dilithium.

Para realização das medidas foi utilizado um computador com processador Intel Core i5-8265U, CPU 1.6 GHz, OS Ubuntu 20.04.6 LTS e memória RAM de 8 Gb. Utilizamos o software Massif, disponível no programa de avaliação Valgrind 3.15.0, como ferramenta para medição da memória RAM usada em cada função. Para a medição de ciclos de CPU, utilizamos o programa perf, calculando a média e a mediana de ciclos em cada função. As medianas são mostradas, dada a presença de alguns outliers (em particular na função Sign) devido ao cálculo de *rejection sampling*.

5. Resultados e discussões

Os resultados obtidos são apresentados nas Tabelas 1, 2 e 3, para os picos de consumo de RAM em cada função e os valores de média e mediana de ciclos de CPU consumidos para as versões padrão (v.0) e otimizada com TurboSHAKE (v.1). A partir da otimização implementada, foi possível constatar que não houve impacto no consumo de memória (como já esperado), uma vez que o algoritmo de hash otimizado tem a mesma estrutura do original. No tocante ao consumo de CPU, notamos uma redução para todos os níveis de segurança, o que se justifica pela grande utilização de funções hash no Dilithium, especialmente na geração de chaves e verificação de assinaturas. Há também uma maior economia em termos de ciclos de CPU para níveis de segurança mais altos, já que o Dilithium trabalha com parâmetros maiores nesses níveis, e exige mais das funções hash. Apesar dos números de ciclos apresentados aqui serem ligeiramente superiores aos mostrados no site oficial do projeto Dilithium [1] devido às diferenças entre processadores, pode ser inferido que também serão obtidos os mesmos níveis de redução de ciclos pela mudança da funções hash em outras implementações.

Tabela 1. Uso máximo de RAM (bytes) para níveis de segurança 2, 3 e 5

Pico de RAM para CRYSTALS-Dilithium v.0 / v.1			
Nível de segurança	KeyGen	Sign	Verify
Dilithium 2	42.488 B	59.648 B	43.936 B
Dilithium 3	67.128 B	90.272 B	68.344 B
Dilithium 5	105.496 B	134.960 B	106.448 B

6. Conclusões e trabalhos futuros

A otimização das funções de hash utilizadas no algoritmo de assinatura pós-quântica CRYSTALS-Dilithium causou um impacto significativo no desempenho do mesmo, sem

[1] https://pq-crystals.org/dilithium/index.shtml

Tabela 2. Média de ciclos de CPU para cada função do Dilithium

Média de ciclos de CPU para o CRYSTALS-Dilithium			
Nível de segurança	Função	v.0	v.1 (variação %)
Dilithium 2	KeyGen	418.522	325.304 (-22.3%)
	Sign	1.844.839	1.674.893 (-9.2%)
	Verify	447.904	362.306 (-19.1%)
Dilithium 3	KeyGen	740.820	566.839 (-23.4%)
	Sign	2.781.223	2.525.570 (-9.2%)
	Verify	705.726	556.641 (-21.1%)
Dilithium 5	KeyGen	1.144.005	842.012 (-26.4%)
	Sign	3.523.184	3.255.138 (-7.6%)
	Verify	1.173.812	899.562 (-23.3%)

Tabela 3. Mediana de ciclos de CPU para cada função do Dilithium

Mediana de ciclos de CPU para o CRYSTALS-Dilithium			
Nível de segurança	Função	v.0	v.1 (variação %)
Dilithium 2	KeyGen	411.942	321.624 (-21.9%)
	Sign	1.490.671	1.279.631 (-14.2%)
	Verify	446.843	361.137 (-19.2%)
Dilithium 3	KeyGen	724.807	562.345 (-22.4%)
	Sign	2.230.749	2.080.019 (-6.8%)
	Verify	704.122	553.911 (-21.3%)
Dilithium 5	KeyGen	1.119.261	836.480 (-25.3%)
	Sign	3.026.118	2.665.533 (-11.9%)
	Verify	1.168.728	897.902 (-23.2%)

diminuir sua segurança. Como trabalhos futuros, temos a otimização do custoso cálculo de NTT, buscando um compromisso entre memória e processamento, e a busca por mais reduções no uso de RAM, a fim de facilitar a implementação em dispositivos com recursos limitados sem comprometer a segurança.

Referências

Aumasson, J. P. (2019). Too much crypto. Cryptology ePrint Archive, Paper 2019/1492. `https://eprint.iacr.org/2019/1492`.

Bertoni, G., Daemen, J., Hoffert, S., Peeters, M., Assche, G. V., Keer, R. V., and Viguier, B. (2023). Turboshake. Cryptology ePrint Archive, Paper 2023/342.

NIST (2022). Nist announces first four quantum-resistant cryptographic algorithms. `https://www.nist.gov/news-events/news`. Acessado em 22/06/2022.

Shor, P. W. (1997). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Journal on Computing.