Aplicação de esquemas de assinatura digital pós-quântica baseada em reticulados para hardwares restritos

Rodrigo Duarte de Meneses , Marco Aurélio Amaral Henriques
{r197962@dac.unicamp.br, maah@dca.fee.unicamp.br}

Departamento de Engenharia de Computação e Automação (DCA)
Faculdade de Engenharia Elétrica e de Computação (FEEC)
Universidade Estadual de Campinas (Unicamp)
Campinas, SP, Brasil

Abstract – Neste artigo, apresentamos o esquema de assinatura digital pós-quântica Dilithium, parte da suíte CRYSTALS (Cryptographic Suite for Algebraic Lattices), composta por algoritmos criptográficos baseados em reticulados. O Dilithium é um dos candidatos ao processo de padronização de algoritmos pós-quânticos do NIST (National Institute of Standards and Technology), e possui grande relevância em aplicações no contexto de ambientes computacionais restritos. As tentativas de otimização dos recursos computacionais do Dilithium para viabilizar sua implementação em hardwares restritos comumente buscam aprimorar os cálculos associados à NTT. Entretanto, a implementação do Dilithium com funções de hash criptográfico alternativas constitui uma opção válida para potenciais otimizações do algoritmo. Este trabalho busca analisar o impacto das funções de hash nos requisitos de processamento do Dilithium para sua posterior aplicação em ambientes computacionais restritos.

Keywords – criptografia pós-quântica, criptografia baseada em reticulados, assinaturas digitais

1. Introdução

1.1. Motivações e relevância

De acordo com os padrões vigentes de criptografia de chave simétrica, os criptossistemas mais populares são baseados no RSA ou em curvas elípticas. A segurança desses esquemas é associada à dificuldade do problema do logaritmo discreto e de fatoração de primos.

Com o advento da computação quântica, a viabilização de algoritmos como o algoritmo de Shor [12] mostra que esses problemas podem ser resolvidos em tempo polinomial [11]. Com isso, a segurança atribuída aos criptossistemas mais amplamente utilizados foi comprometida. O desafio de desenvolvimento de esquemas de encriptação resistentes aos ataques de um computador quântico configura a área da criptografia pós-quântica.

Em 2016, o NIST anunciou um processo de padronização de algoritmos pós-quânticos [7]. Dentre os algoritmos finalistas do terceiro round, destacam-se os algoritmos de assinatura digital pós-quântica baseados em reticulados – FALCON e CRYSTALS-Dilithium.

O FALCON utiliza-se do criptossistema NTRU e apresenta a maior velocidade de verificação e também os menores tamanhos de chave necessários [8]. Entretanto, a utilização do criptossistema NTRU constitui uma dificuldade para sua implementação em ambientes computacionais restritos. O Dilithium, por sua vez, não depende do criptossistema NTRU para o seu funcionamento e pode ser melhor aplicado no contexto de hardwares restritos.

1.2. Princípios de funcionamento

O problema básico sob o qual o Dilithium adquire sua segurança é o problema MLWE (Module Learning With Errors) [10], esquematizado na Figura 1.

O MLWE consiste em uma matriz $\mathbf{A}$ e um conjunto de vetores $\mathbf{t}$, $\mathbf{s_1}$ e $\mathbf{s_2}$ com seus elementos em um anel polinomial R. Esse problema pode ser descrito como a dificuldade de distinguir um vetor qualquer $\mathbf{t} \in R^k$ de um vetor da forma $\mathbf{t} = \mathbf{A} \cdot \mathbf{s_1} + \mathbf{s_2}$. Esse problema é uma generalização do problema LWE (Learning With Errors), onde tomamos $R = \mathbb{Z}_q$. Os problemas MLWE e LWE são considerados difíceis [9] [6].

$$\mathbf{t} = \begin{pmatrix} t_1 \\ \vdots \\ t_k \end{pmatrix} = \underbrace{\begin{pmatrix} a_{1,1} & \dots & a_{1,l} \\ \vdots & \ddots & \vdots \\ a_{k,1} & \dots & a_{k,l} \end{pmatrix}}_{\text{uniform, public}} \underbrace{\begin{pmatrix} s_{1,1} \\ \vdots \\ s_{1,l} \end{pmatrix}}_{\text{short}} + \underbrace{\begin{pmatrix} s_{2,1} \\ \vdots \\ s_{2,k} \end{pmatrix}}_{\text{short}}$$

Figura 1. Esquema básico do problema MLWE.

No Dilithium, a matriz $\mathbf{A}$ e o vetor $\mathbf{t}$ são tomados como parâmetros públicos e os vetores $\mathbf{s_1}$ e $\mathbf{s_2}$ constituem a chave privada. O vetor $\mathbf{s_2}$ é também chamado de vetor de erros, por ser responsável por incluir os ruídos no cálculo de $\mathbf{t}$. O anel polinomial utilizado no esquema é da forma $R = \mathbb{Z}_q[X]/(X^n + 1)$, com $q = 8.380.417$ e $n = 256$.

A maior dificuldade, portanto, advém do armazenamento dos parâmetros $\mathbf{A}$, $\mathbf{t}$, $\mathbf{s_1}$ e $\mathbf{s_2}$. Por se tratarem de matrizes e vetores de polinômios, o armazenamento desses parâmetros ocupa muito espaço de memória. A solução utilizada é gerar os parâmetros a partir de uma seed ρ através de uma função de hash criptográfico. Com isso, é necessário que armazenemos apenas a seed ρ, e os parâmetros são gerados conforme a demanda.

Para esse propósito são empregadas as rotinas ExpandA, ExpandS e ExpandMask, chamadas de funções de expansão. Essas funções são responsáveis por gerar os parâmetros necessários a partir de uma seed aleatória ρ.

1.3. Trabalhos relacionados

As tentativas de otimização do Dilithium para implementação em ambientes computacionais restritos [5] [3] costumam ter como objetivo aumentar sua eficiência através de otimizações no cálculo da NTT (Number Theoretic Transform), uma forma particular da transformada discreta de Fourier para corpos finitos. Essas otimizações são importantes no sentido de agilizar as multiplicações entre polinômios, um dos processos de maior custo computacional no Dilithium [4]. No entanto, as otimizações propostas não aumentam a eficiência das funções de expansão (ExpandA, ExpandS e ExpandMask), utilizadas em todo algoritmo.

Diante disso, esta pesquisa busca analisar a possibilidade implementação do Dilithium com diferentes funções de hash criptográfico e seu desempenho em hardwares restritos.

2. Proposta

Tendo em vista que a utilização das funções de hash criptográfico é um dos aspectos mais computacionalmente custosos no Dilithium [4], é válido analisar como essas funções são utilizadas.

As funções de hash são empregadas majoritariamente nas funções de expansão. Como visto, as funções de expansão são responsáveis por gerar parâmetros do algoritmo a partir de seeds amostradas de uma distribuição uniforme. Na implementação original do Dilithium, são utilizadas as funções SHAKE-128 e SHAKE-256 para o cálculo dos hashes.

Na documentação submetida para o round 3 do NIST [4] é apresentada uma implementação alternativa utilizando o AES-ctr (modo counter) como função de hash. Essa comparação visa explicitar a eficiência do SHAKE em relação ao AES-ctr quando implementados em software.

No entanto, os resultados disponíveis na literatura [1] apontam para um overhead computacional associado a essas funções para o cálculo de hash. Com o intuito de implementação em ambientes restritos, a utilização de funções de hash com menores requisitos de memória e processamento consitui uma possível complementação às otimizações nos cálculos da NTT.

3. Resultados

A partir da proposta de otimização através das funções de hash criptográfico, buscamos determinar se o impacto dessas funções é de fato significativo nos requisitos de memória e processamento do Dilithium. Fazemos esta análise determinando a contribuição percentual das funções de expansão (que correspondem a maior parte da utilização das funções de hash) nos ciclos de processador utilizados para as principais rotinas do Dilithium.

A Tabela 1 mostra a média e mediana dos ciclos de processador para cada uma das três funções de expansão utilizadas, calculadas para 100.000 iterações do algoritmo. Para cada iteração é feita a geração de chaves, assinatura de uma mensagem aleatória de 59 bytes e verificação dessa assinatura. A decisão de apresentar também a mediana foi feita devido à presença de alguns valores atípicos (outliers), fenômeno esperado por conta da utilização de rejection sampling no Dilithium.

Adotamos o set de parâmetros 3 indicado na documentação [4], e todas as medidas foram feitas por um laptop com processador Intel Core i5-8265U CPU 1.6 GHz, SO Ubuntu 20.04.4 LTS e memória RAM de 8 Gb.

-	ExpandA	ExpandS	ExpandMask
Média	161.088	2.718	4.670
Mediana	145.069	2.398	4.566

Tabela 1. Medidas dos ciclos de processador para as funções de expansão.

Na Tabela 2 são apresentados os valores de média e mediana das três funções principais do Dilithium: KeyGen (geração de chaves pública e privada), Sign (procedimento de assinatura) e Verify (verificação de assinatura).

-	KeyGen	Sign	Verify
Média	305.068	1.236.360	304.456
Mediana	299.730	992.632	296.411

Tabela 2. Medidas dos ciclos de processador para as funções principais.

A partir da frequência com que cada uma das funções de expansão é utilizada nas rotinas principais do Di-

lithium, pode-se analisar a contribuição de cada uma das funções de expansão para as rotinas principais, da perspectiva da média de ciclos de processador. Isso é feito calculando a razão entre os ciclos de processador de cada uma das funções de expansão em relação às rotinas principais. Por fim, somamos as contribuições das funções de expansão de forma a obter a sua contribuição geral para cada uma das rotinas. Os resultados obtidos são expressos na Tabela 3.

-	KeyGen	Sign	Verify
ExpandA	52.8%	13.0%	52.9%
ExpandS	0.8%	0%	0%
ExpandMask	0%	0.3%	0%
Σ	53.6%	13.3%	52.9%

Tabela 3. Percentual de contribuição de cada uma das funções de expansão para a média de ciclos de processador das funções principais.

Percebemos, da Tabela 3, que as funções de expansão correspondem, em média, a mais de metade dos ciclos de processador utilizados para as funções de geração de chaves e verificação de assinatura. Esse resultado respalda a hipótese de que as funções de hash possuem um impacto substancial nos requisitos de processamento do algoritmo.

4. Conclusões

A fim de viabilizar a aplicação do Dilithium em ambientes computacionais restritos, buscou-se determinar quais rotinas do algoritmo são responsáveis pelo maior percentual de processamento realizado. Conforme indicado pelos resultados, as funções de expansão correspondem à grande parte dos ciclos de processador. Assim, indica-se a otimização das funções de hash como uma potencial abordagem no sentido de possibilitar a aplicação do Dilithium em hardwares restritos.

Com a possibilidade de otimização do Dilithium a partir de funções de hash criptográfico alternativas, pode-se apontar como posteriores etapas desta pesquisa a implementação do Dilithium com as funções de hash da família BLAKE [2], ou o KangarooTwelve, que aparentam ser opções com menor custo computacional para o cálculo de hashes. A partir dessas possíveis otimizações, podemos buscar a aplicação do Dilithium em ambientes computacionais restritos, tais quais os microcontroladores ARM Cortex M0+ e Cortex M4.

Agradecimentos

Agradecemos ao suporte do Research Group of Applied Security (ReGrAS), da Faculdade de Engenharia Elétrica e Computação (FEEC), para elaboração da pesquisa, bem como para as discussões que se provaram imprescindíveis em sua produção.

Referências

[1] Jean-Philippe Aumasson. Too much crypto. *Real-World Crypto*, May 2020.

[2] Jean-Philippe Aumasson et al. Sha-3 proposal blake. `https://decred.org/research/aumasson2010.pdf`, 2010. (acessado em 08/08/2022).

[3] Luke Beckwith et al. High-performance hardware implementation of crystals-dilithium. *2021 International Conference on Field-Programmable Technology (ICFPT)*, November 2021.

[4] Vadim Lyubashevsky et al. Crystals-dilithium: Algorithm specifications and supporting documentation. `https://pq-crystals.org/dilithium/`, 2021. (acessado em 08/08/2022).

[5] Youngbeom Kim et al. Crystals-dilithium on armv8. *Hindawi Security and Communication Networks*, 2022.

[6] Daniele Micciancio. *Complexity of Lattice Problems*. Kluwer Academic, 2002.

[7] National Institute of Standards and Technology. Request for comments on post-quantum cryptography requirements and evaluation criteria. `https://csrc.nist.gov/projects/post-quantum-cryptography`, 2016. (acessado em 08/08/2022).

[8] Thomas Prest. 3rd round update on the falcon candidate algorithm. `https://csrc.nist.gov/Presentations/2021/falcon-round-3-presentation`, 2021. (acessado em 08/08/2022).

[9] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 2009.

[10] Gregor Seiler. Rystals-dilithium: A lattice-based digital signature scheme. *Conference on Cryptographic Hardware and Embedded Systems*, September 2018.

[11] Peter W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, November 1994.

[12] Peter W. Shor. Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 1997.