Redução do consumo de memória no algoritmo de criptografia pós-quântica SABER

George Gigilas Junior, Marco Aurélio Amaral Henriques

{georgejuniorg@yahoo.com.br, marco@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 – SABER é um algoritmo pós-quântico IND-CCA2 seguro de troca de chaves, finalista da terceira rodada do processo de padronização de algoritmos pós-quânticos do NIST. Visando tornar sua execução eficiente em ambientes computacionais restritos, este trabalho implementa, em um microcontrolador ARM Cortex-M0+, otimizações da literatura de outras arquiteturas para reduzir o consumo de memória de pilha desse algoritmo e de suas variações LightSABER e FireSABER. Considerando a versão de segurança média, foi possível reduzir o consumo de memória de pilha de suas funções entre 18% e 39%, comparado com resultados para ARM Cortex-M0 disponíveis na literatura. Ademais, descobriu-se que a flag de compilação -O1 produziu os melhores resultados para a versão de menor segurança, e que a flag -O2 foi a melhor para as demais versões.

Keywords – Criptografia pós-quântica, ARM Cortex M0+, SABER, reticulados.

1. Introdução

Com os avanços no desenvolvimento de computadores quânticos, um poder computacional ainda maior se aproxima. Apesar de promissor para diversas pesquisas, ele também se mostra uma ameaça para a segurança de diversas aplicações na internet. Já existem algoritmos quânticos capazes de quebrar os esquemas de criptografia assimétrica atuais, baseados em fatoração de números primos ou em logaritmos discretos.

Visando algoritmos que permaneçam seguros após o advento dessa tecnologia, o órgão americano National Institute of Standards and Technology (NIST) promoveu um processo de padronização de algoritmos pós-quânticos, iniciado em 2016 [9]. Ele é composto por algumas rodadas em que os melhores algoritmos são selecionados e outros descartados, tanto para algoritmos de troca de chaves quanto para algoritmos de assinatura digital. Em 2021, eram quatro os candidatos de troca de chaves na terceira rodada, sendo três deles baseados em problemas com reticulados, o que tornava muito provável que algum algoritmo baseado em reticulados fosse escolhido.

Porém, os problemas com reticulados são pesados computacionalmente. As chaves desses esquemas são bastante grandes, o que faz com que as operações fiquem mais caras. Diante disso, foram propostas várias otimizações na literatura para viabilizar os algoritmos em ambientes computacionais restritos, que podem apresentar dificuldades para executá-los pela quantidade limitada de memória disponível.

Uma vez que dispositivos IoT (Internet of Things), que possuem recursos limitados, estão cada vez mais presentes no dia-a-dia, é importante que eles também sejam protegidos. Com base nisso, decidimos buscar otimizar o consumo de memória do algoritmo SABER [4], que ainda era finalista quando a pesquisa foi iniciada. Também foram avaliados os requisitos de memória para as variações LightSABER e FireSABER. A primeira é uma versão mais leve e menos segura, enquanto a segunda é mais robusta e mais segura. Por fim, fizemos diversos testes para determinar a melhor flag de otimização para cada versão do algoritmo.

Vale notar que, apesar de não ter sido escolhido no processo do NIST, o SABER é bastante similar ao algoritmo escolhido (CRYSTALS-KYBER [2]), o que possivelmente permite um intercâmbio de técnicas de otimização entre os dois algoritmos. Além disso, ele se mostrou bastante eficiente em ambientes restritos, o que pode ser bastante importante em alguns contextos.

1.1 SABER

O SABER é um mecanismo de encapsulamento de chaves (KEM) pós-quântico que, como todo KEM, faz a geração de chaves, o encapsulamento e o desencapsulamento de mensagens (de 32 bytes, que são usadas como chaves de sessão em algoritmos de criptografia simétrica). Ele é IND-CCA2 seguro [1], o que significa que nenhum adversário possui vantagem significativa para conseguir distinguir pares de texto cifrado baseados nas mensagens cifradas por ele.

Como citado anteriormente, a segurança do algoritmo se baseia em problemas com reticulados, mais especificamente o Module Learning with Rounding (Module-LWR). Tal problema corresponde a uma variação do Learning with Errors (LWE) em que,

ao invés de inteiros, tem-se polinômios nas entradas das matrizes e os erros (ou ruídos) são calculados de forma determinística. Já o LWE, se trata do problema de resolver equações matriciais (correspondentes a combinações lineares de vetores) acrescidas de ruídos, como ilustrado na Figura 1. A matriz **A** e o vetor **b** formam a chave pública, o vetor **s** é a chave privada, o vetor **e** é composto pelos ruídos e o inteiro *p* define o conjunto em que as operações são realizadas [10].

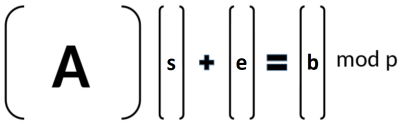

Figura 1. Esquematização do LWE.

1.2 LightSABER e FireSABER

O LightSABER e o FireSABER, como citado anteriormente, são variações oficiais do SABER, com mudança em alguns parâmetros que possibilitam níveis de segurança e de complexidade distintos. Entre os parâmetros que se diferem, o principal é o número de polinômios no vetor secreto e na matriz pública, sendo o segundo correspondente ao quadrado do primeiro. Dessa forma, o LightSABER possui 2 polinômios no vetor **s** (4 na matriz **A**), enquanto o vetor **s** no SABER conta com 3 polinômios (9 na matriz **A**) e o FireSABER possui 4 polinômios no vetor **s** (16 na matriz **A**). Por consequência, as três versões possuem tamanho de chaves e nível de segurança distintos: a versão mais leve possui chave pública de 672 bytes, chave privada de 1.568 bytes e nível de segurança quântica de 115; a versão original possui chave pública de 992 bytes, chave privada de 2.304 bytes e nível de segurança quântica de 180; a versão mais robusta possui chave pública de 1.312 bytes, chave privada de 3.040 bytes e nível de segurança quântica de 245.

2. Proposta

Diante do contexto de criptografia pós-quântica, a pesquisa tem como proposta reduzir os requisitos de memória de pelo menos um dos algoritmos submetidos ao NIST, para serem executados de forma eficiente em ambientes computacionais restritos. Ao longo dos estudos, optou-se por estudar o SABER e suas variações e executá-los em um microcontrolador ARM Cortex-M0+, dispositivo bastante limitado.

Para tanto, estudamos otimizações propostas na literatura para implementações em ARM Cortex-M0 e Cortex-M4. Vale ressaltar que as otimizações são voltadas para reduzir o consumo de memória de pilha, memória extra alocada ao longo da execução do programa. A memória de programa, por sua vez, é bastante abundante e é mais do que suficiente, portanto ela não é alvo de otimizações. Quanto às variáveis globais, elas armazenam variáveis importantes que não podem ser reduzidas.

Com relação às diferenças entre as arquiteturas ARM Cortex-M0 e Cortex-M0+, a mudança do pipeline de três para dois estágios pode impactar a contagem de ciclos. Apesar de a frequência de *clock* ter aumentado, ela não impacta os resultados pois foi medido o número de ciclos de *clock*. De forma geral, ambas são bastante parecidas e o consumo de memória deve ser bem próximo para as duas arquiteturas.

2.1 Otimizações propostas na literatura

Como já foi visto, a segurança e a complexidade do algoritmo é bastante relacionada às operações com polinômios, que são bastante custosas, especialmente porque todos eles têm 256 coeficientes de 13 bits. Por esse motivo, grande parte das otimizações propostas na literatura são feitas acerca disso. Em todo o esquema, a multiplicação de polinômios é a que mais consome memória de pilha e, como o módulo da aritmética modular do SABER não é uma potência de dois (é um número primo), ele não utiliza a Number Theoretic Transform (NTT) [5], que é a forma mais eficiente de fazer tal operação. Alternativamente, utiliza-se uma combinação dos algoritmos de Toom-Cook [3] e Karatsuba [3], que é a forma compatível mais eficiente. O problema é que esses algoritmos consomem memória de pilha da ordem de O(n), já que são recursivos e requerem memória adicional (não são *in-place*).

Em troca de velocidade, Karmakar et. al [7] propuseram uma versão *in-place* do algoritmo de Karatsuba que consome O(log n) de memória de pilha para as multiplicações, sendo vantajoso para ambientes restritos em memória. Além disso, Karmakar et. al propõem uma estratégia *just-in-time* para gerar a matriz **A** e o vetor secreto **s**. A partir dela, os polinômios são gerados um por vez, reaproveitando seu espaço na memória, de acordo com a demanda nas operações. Dessa forma, apesar de as operações ficarem mais lentas, pela necessidade de gerar os polinômios várias vezes, o consumo de memória de pilha reduz bastante. Foi necessário adaptar essas otimizações para o LightSABER e para o FireSABER, já que Karmakar et. al implementaram-nas apenas para o SABER.

Além dessas otimizações, estudamos propostas já desenvolvidas para dispositivos ARM Cortex-M4, buscando trazer as que não dependem de instruções específicas para o ARM Cortex-M0+. Dentre as propostas da literatura [8], foram implementadas duas delas. A primeira corresponde à codificar os coeficientes dos polinômios do vetor secreto **s** com apenas 4 bits, ao invés de 13 bits. A literatura afirma que essa mudança não impacta a segurança do algoritmo, por não utilizar o NTT. Adicionalmente, essa

mudança torna as funções de empacotamento (que compactam os coeficientes dos polinômios) mais simples, aumentando o desempenho do algoritmo. A segunda proposta consiste em utilizar uma versão *in-place* da verificação do texto cifrado (durante a decifração), reduzindo o uso de memória de pilha.

3. Resultados

Aplicando as otimizações descritas, o código resultante foi executado na placa de desenvolvimento FRDM-KL25Z, que possui MCU KL25Z128 (processador ARM Cortex-M0+), 128 kB de memória flash, 16 kB de memória SRAM e 48 MHz de frequência de clock. A IDE MCUXpresso, desenvolvida pela NXP, foi utilizada para a realização dos testes, já que ela possui uma ferramenta integrada para medição de uso de memória de pilha. Essa IDE possui um compilador embutido GNU Arm Embedded Toolchain 2021.07. Para contar o número de ciclos de *clock*, utilizou-se a interface CMSIS (Cortex Micro-Controller Software Interface and Standard).

Algoritmo		Geração de chaves (variação %)	Encapsulamento (variação %)	Desencapsulamento (variação %)
SABER (-O2)	Memória (kB)	4,13	3,75	3,77
	Ciclos	4.495.576	5.940.149	6.930.342
LightSABER (-O1)	Memória (kB)	3,36 (-19%)	3,46 (-8%)	3,49 (-7%)
	Ciclos	2.172.163 (-52%)	3.157.820 (-47%)	3.815.365 (-45%)
FireSABER (-O2)	Memória (kB)	4,88 (+18%)	4,00 (+7%)	4,02 (+7%)
	Ciclos	7.743.499 (+72%)	9.630.721 (+62%)	10.973.725 (+58%)

Tabela 1. Consumo de memória de pilha e ciclos de clock dos algoritmos na melhor flag de compilação.

Função		SABER em M0 [7]	SABER em M4 *speed/memory* [1] [6]	SABER em M4 *speed* [2] [6]	Este trabalho
Geração de chaves	Memória (kB)	5,03	3,79 (-25%)	6,64 (+32%)	4,13 (-18%)
	Ciclos (mil)	4.786	820 (-83%)	645 (-87%)	4.495 (-6%)
Encapsulamento	Memória (kB)	5,12	3,18 (-38%)	7,32 (+43%)	3,75 (-27%)
	Ciclos (mil)	6.328	1.059 (-83%)	851 (-87%)	5.940 (-6%)
Desencapsulamento	Memória (kB)	6,22	3,19 (-49%)	7,32 (+18%)	3,77 (-39%)
	Ciclos (mil)	7.509	1.038 (-86%)	774 (-90%)	6.930 (-8%)

[1] Versão dedicada a otimização de memória de pilha, mas também levando em conta a velocidade.
[2] Versão dedicada a otimização de velocidade de execução.

Tabela 2. Resultados dos melhores testes do SABER comparados com os valores de referência.

Quanto à compilação, fizemos testes para diversas flags de otimização, para as três versões do algoritmo, comparando ciclos de clock e memória de pilha. Para o LightSABER, a flag -O1 produziu resultados melhores. Já para as outras versões, a flag com melhor custo-benefício foi a -O2. Ao contrário do esperado, a flag -O3 não se comportou tão bem, o que sugere que ela deve ser utilizada com cautela ou até evitada. Na Tabela 1, estão dispostos os melhores valores de ciclos de clock e de memória de pilha encontrados. Como esperado pela quantidade de operações, os ciclos de clock variaram bastante conforme a versão do algoritmo. Já o consumo de memória de pilha se manteve baixo nas três versões.

Para comparação, utilizamos os valores disponibilizados no oficial do algoritmo e pelo PQM4 (*framework* responsável por bibliotecas, avaliação e

testes com algoritmos de criptografia pós-quântica [6]). Porém, não foram disponibilizados dados para o LightSABER e para o FireSABER em Cortex-M0, impossibilitando tal comparação. Os dados comparativos se encontram na Tabela 2 e indicam uma melhora considerável em relação à implementação em Cortex-M0, provavelmente por conta das otimizações trazidas da implementação para Cortex-M4. Ainda sobre a Tabela 2, nota-se que, como esperado, a implementação em Cortex-M4 apresentou melhores resultados, pelo seu poder computacional e por ter mais otimizações feitas para essa arquitetura. Mesmo assim, os resultados deste trabalho são importantes pois apresentaram uma grande redução de memória de pilha e ainda uma redução de ciclos de clock. Destacamos o menor gasto de memória de pilha, comparado com a versão focada em velocidade, e o consumo de memória de pilha similar ao da versão focada em memória.

4. Conclusões

Com a finalidade de diminuir os requisitos de memória do algoritmo SABER (e suas variações) e torná-lo eficiente em ambientes computacionais restritos, estudamos e implementamos diversas otimizações da literatura para serem executadas em um dispositivo ARM Cortex-M0+. Ademais, fizemos testes com várias flags de otimização na compilação para determinar a melhor delas para cada algoritmo. Concluímos que a flag -O1 mostrou os melhores resultados para o LightSABER e, a flag -O2, para as demais versões.

Comparando os resultados obtidos com os resultados para Cortex-M0 disponíveis na literatura [7], conseguimos reduzir entre 6% e 8% o número de ciclos de clock e entre 18% e 39% o uso de memória. Embora não tenha resultados nessa plataforma para as variações do SABER, espera-se que os valores sejam similares e proporcionais, considerando as diferenças entre as versões. Também comparamos com resultados de memória obtidos para Cortex-M4 e chegamos próximo dos resultados da versão otimizada em memória e até superar os resultados da versão que foca em velocidade.

A partir deste estudo, pode-se continuar a trazer otimizações desenvolvidas para outras plataformas, visando principalmente otimizar a velocidade (sem comprometer o consumo de memória), uma vez que o consumo de memória já foi bastante reduzido. Para trabalhos futuros, propomos uma otimização mais detalhada a nível de linguagem assembly, podendo melhorar ainda mais o desempenho do algoritmo.

Agradecimentos

Agradecemos aos membros do grupo ReGrAS (Research Group on Applied Security), que participaram e contribuíram para discussões acerca dos estudos desenvolvidos nesta pesquisa, e ao Programa Institucional de Bolsas de Iniciação Científica (PIBIC) do CNPq e à Pró-Reitoria de Pesquisa da Unicamp, pela oportunidade e suporte da pesquisa.

Referências

[1] Bogdanov, D. (2005), “IND-CCA2 secure cryptosystems”, University of Tartu.

[2] Bos, J., Ducas, et. al (2018), “CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM”, 2018 IEEE European Symposium on Security and Privacy, EuroS&P.

[3] Crandall, R., Pomerance, C. (2005), “Prime Numbers – A Computational Perspective”, Second Edition, Springer, Section 9.5.1: Karatsuba and Toom–Cook methods, p.473.

[4] D’Anvers, JP., et. al (2018), “Saber: Module-LWR Based Key Exchange, CPA-Secure Encryption and CCA-Secure KEM”, In: Joux A., Nitaj A., Rachidi T. (eds) Progress in Cryptology - AFRICACRYPT 2018 - 10th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 7-9, 2018, Proceedings. Lecture Notes in Computer Science 10831, Springer 2018, ISBN 978-3-319-89338-9.

[5] Hedge, S., Nagapadma, R. (2019), “Number Theoretic Transform for Fast Digital Computation”, Department of Electronic and Communication Engineering, NIE Institute of Technology.

[6] Kannwischer, M. et. al (2019), “PQM4: Post-quantum crypto library for the ARM Cortex-M4”, https://github.com/mupq/pqm4. (acessado em 27/06/2022)

[7] Karmakar, A., et. al (2018), “Saber on ARM. CCA-secure module lattice-based key encapsulation on ARM”, In Transactions in Cryptographic Hardware and Embedded Systems.

[8] Mera, J., et. al (2020), “Time-memory trade-off in Toom-Cook multiplication: an application to module-lattice based cryptography”, In Transactions in Cryptographic Hardware and Embedded Systems.

[9] National Institute of Standard and Technology – NIST (2016), “Request for Comments on Post-Quantum Cryptography Requirements and Evaluation Criteria”, Notice 81 FR 50686, p. 50686-50687.
https://csrc.nist.gov/projects/post-quantum-cryptography (acessado em 09/08/2022)

[10] Regev, O. (2005), “The Learning with Errors Problem”, Courant Institute of Mathematical Sciences, New York University.