Criptografia pós-quântica na comunicação IPC sobre o microkernel seL4

Elias Matos Garcia - elias.m@edu.pucrs.br
Fabiano Passuelo Hessel - fabiano.hessel@pucrs.br

Julho, 2024

Resumo

Este trabalho investiga a integração de algoritmos de criptografia pós-quântica utilizando a biblioteca liboqs no microkernel seL4. O objetivo é modificar as primitivas de comunicação do microkernel para assegurar a comunicação segura entre processos (IPC), utilizando os algoritmos Crystals-Kyber e Crystals-Dilithium para troca de chaves e assinaturas digitais, respectivamente. A proposta é analisada em termos de segurança, desempenho e viabilidade em sistemas embarcados críticos, proporcionando uma solução robusta contra as ameaças emergentes da computação quântica.

1 Introdução

Com os avanços rápidos na computação quântica, surgem preocupações significativas sobre a segurança dos sistemas criptográficos tradicionais, que podem ser facilmente quebrados por computadores quânticos. Em resposta a essas ameaças, algoritmos de criptografia pós-quântica têm sido desenvolvidos para garantir a segurança em um cenário onde computadores quânticos são uma realidade.

O microkernel seL4, conhecido por sua segurança e corretude formal, é amplamente utilizado em sistemas embarcados críticos. No entanto, a comunicação interprocessual (IPC) dentro desse microkernel ainda precisa de soluções que garantam a integridade e confidencialidade das mensagens trocadas, especialmente contra ataques quânticos, eavesdropping, falsificação de mensagens e ataques de man-in-the-middle.

Este trabalho propõe a modificação das primitivas de comunicação do seL4 para incorporar algoritmos de criptografia pós-quântica, utilizando a biblioteca liboqs. O sistema utilizará os algoritmos Crystals-Kyber e Crystals-Dilithium, padronizados pelo NIST, para troca de chaves e assinaturas digitais, respectivamente. A solução visa proporcionar um canal de comunicação seguro entre processos, garantindo a proteção contra ataques quânticos e assegurando a autenticidade das mensagens trocadas.

A estrutura deste trabalho é a seguinte: inicialmente, discutiremos os fundamentos da criptografia pós-quântica e os algoritmos selecionados. Em seguida, apresentaremos o microkernel seL4 e descreveremos a arquitetura proposta para a modificação das primitivas de comunicação. Finalmente, apresentaremos os resultados e conclusões desta pesquisa.

2 Criptografia pós-quântica

O rápido avanço da computação quântica tem levantado preocupações de órgãos de extrema importância na área da cibersegurança, como o National Institute of Standards and Technology (NIST) e National Security Agency (NSA), por conseguirem resolver os problemas que garantem funcionamento da maior parte da criptografia atual em tempo computacional bastante reduzido [15].

Em 2016 o NIST abriu espaço para que autores submetessem propostas de algoritmos resistentes a ataques de computadores quânticos e também já discutia a padronização destes, que foram chamados de algoritmos

de criptografia pós-quântica [2]. A meta era estabelecer a padronização de ao menos um algoritmo de criptografia de chave pública, um de assinatura digital e um para troca de chaves. E, em 2022, entre os algoritmos submetidos foram selecionados 4 [10], para troca de chaves e criptografia de chave pública o Crystals-Kyber e para assinatura digital Crystals-Dilithium, Falcon e SPHINCS+.

Para o escopo deste trabalho se mostrou necessário um algoritmo para estabelecimento de um canal de comunicação seguro entre duas aplicações e para fazer a criptografia das mensagens trocadas com intuito de evitar *eavesdropping* e um algoritmo para assinar as mensagens que serão trocadas para garantir sua autenticidade. Com esses requisitos as escolhas foram do Krystals-Kyber, única opção dos algoritmos padronizados pelo NIST, e o Crystals-Dilithium que se mostra muito mais eficiente que os demais [12].

2.1 Crystals-Dilithium

O Crystals-Dilithium é um algoritmo de criptografia pós-quântico de assinatura digital cujo funcionamento se baseia no problema de Shortest Vector Problem (SVP) em *lattices* [3]. O algoritmo é dividido em uma fase de geração de chaves, um processo de assinatura e um de verificação. Na primeira fase pode ser observada no Algoritmo 1, de geração de chave, o algoritmo gera uma matriz A com dimensões k e l, depois gera dois vetores, s e e, pseudo-aleatórios. Por fim, b é calculado gerando a chave pública pk e a *secret key* s.

Algorithm 1 Geração de chaves Crystals-Dilithium

Require: $none$
1: Generate the matrix $A \in R_q^{k \times l}$
2: Sample $s \in R_q^l$ and $e \in R_q^k$
3: Compute $b = (As + e)_q$
4: Output public key $pk = (A, b)$ and secret key $sk = s$

O processo de assinatura, Algoritmo 2 se inicia com a geração de um vetor pseudo-aleatório $y \in R_q^l$, depois uma amostra Ay é armazenada como w. E então m e w passam por uma função de *hashing* H e formam c. Após isso o algoritmo utilizada uma técnica chamada de *rejection sampling*[4] para remover as dependências estatísticas entre z e a chave s, caso z seja rejeitado, o processo começa do início.

Algorithm 2 Processo de assinatura do Crystals-Dilithium

Require: $sk = s, pk = (A, b), m \in 0, 1^*$
1: **repeat**
2: Sample $y \in R_q^l$
3: Compute $w = (Ay)_q$
4: Compute $c = H(w, m) \in B_{60}$
5: Compute $z = y + cs$
6: **until** z is valid
7: Output $\sigma = (z, c)$

Para verificar a validade das mensagens, Algoritmo 3, é usado o w' no módulo original q para calcular o c' (gerado pela operação de *hashing* H juntamento com m) que é comparado com o c da assinatura recebida, caso sejam iguais é uma comunicação válida e segura.

Algorithm 3 Processo de verificação do Crystals-Dilithium

Require: $pk = (A, b), m \in 0, 1^*, \sigma = (z, c)$
1: **if** z is not within bounds **then**
2: **return** $false$
3: **end if**
4: Compute $w' = Az - cb$
5: Compute $c' = H(w'|m)$
6: **if** $c' == c$ **then**
7: **return** $true$
8: **else**
9: **return** $false$
10: **end if**

2.2 Crystals-Kyber

Como uma solução para estabelecimento seguro de chaves entre duas partes e para a criptografia de mensagens trocadas, o Crystals-Kyber é uma alternativa. Ele é um *Key Encapsulation Mechanism* pós-quântico baseado em *lattices*, mais especificamente no problema *Learning with Errors* (LWE) [1]. O algoritmo usado para a geração de chaves no Kyber é exatamente igual a utilizada no Dilithium, descrita no algoritmo 1.

A comunicação se inicia quando um dos participantes gera um chave pública e privada, Algoritmo 1, e envia a chave pública para outro participante que a encapsula, Algoritmo 4, gerando uma chave compartilhada para si e um texto cifrado que é enviado de volta para o criador da chave que o desencapsula, Algoritmo 5 com sua chave privada para também obter a chave compartilhada e por fim ambos criptografam suas mensagem utilizando a chave compartilhada.

Já o encapsulamento de chave é o processo pelo qual uma chave simétrica é gerada e criptografada usando a chave pública do destinatário. Inicialmente, um polinômio r é gerado aleatoriamente. Em seguida, são calculados $u = Ar + e_1 \ mod \ q$ e $v = br + e_2 + mmodq$, onde e_1 e e_2 são vetores de erro gerados pseudo-aleatoriamente e m é a mensagem ou chave simétrica. O texto cifrado é formado por (u, v) e a chave compartilhada é $k = H(v\|m)$, onde H é uma função de *hashing*. O Algoritmo 4 mostra o processo.

Algorithm 4 Encapsulamento Kyber

Require: pk

1: Generate random polynomial r
2: Compute $u = (Ar + e_1)_q$
3: Compute $v = (br + e_2 + m)_q$
4: Output ciphertext $ct = (u, v)$ and shared secret $k = H(v|m)$

O desencapsulamento de chave, mostrado no algoritmo 5, é o processo pelo qual o destinatário usa sua chave secreta para recuperar a chave simétrica encapsulada. O ciphertext ct é dividido em (u, v). Em seguida, calcula-se $m' = v - su \ mod \ q$, onde s é a chave secreta. A chave compartilhada é obtida por $k = H(v|m')$.

Algorithm 5 Desencapsulamento de chave Kyber

Require: sk, ct

1: Parse ct as (u, v)
2: Compute $m' = (v - su)_q$
3: Compute shared secret $k = H(v|m')$

3 seL4

O seL4 é um microkernel, que pode funcionar como um sistema operacional ou um hypervisor, desenvolvido com foco em ser altamente seguro, eficiente e confiável. Ele é o primeiro em ter uma prova formal das suas propriedades [6] o que garante a ele um papel de extrema relevância no cenário de sistemas críticos.

Por ser um microkernel se optou por uma abordagem minimalista, onde só está presente nele o que é estritamente necessário, como mostra a figura 1, tudo que está fora do necessário é executado à modo de usuário [7]. O seL4 possui cerca 9-18k *source lines of code* (SLOC)[13], já um kernel monolítico como o do Linux possui cerca de 22.7M de SLOC onde 16.4M delas (77.9%) são apenas para *device drivers*. Isso tudo faz com que a *Trusted Computing Base* (TCB) seja bastante reduzida.

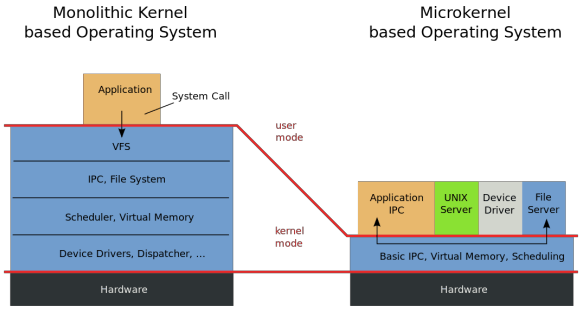

Figure 1: Estrutura de um microkernel comparada a de um kernel monolítico [16].

Para garantir isolamento entre os componentes do sistema e também manter o princípio do *least privilege* o microkernel faz uso de um modelo para controle de acesso chamado *capabilities*. Onde os direitos de acesso, uso de regiões de memória (tanto física quanto virtual), acesso a recursos e dispositivos de hardware podem ser geridos pelo usuário. Em relação a segurança, outros superfícies comuns de ataque como *buffer overflow*, *null pointer dereferences* e *use-after-free* são totalmente evitados por conta de ser um microkernel com corretude formal [8].

3.0.1 Hypervisor

Além de atuar como um sistema operacional ele pode funcionar como um *Hypervisor* de tipo 1 (executado diretamente no harware físico da máquina), fornecendo um ambiente de baixa TCB para virtualização de múltiplas *Virtual Machine Monitors* (VMMs) que podem executar diferentes sistemas operacionais e aplicações. No cenário atual de uma demanda crescente por uso de VMs em sistemas embarcados e possuindo todas as tecnologias que estes requerem (baixo *overhead* de comunicação, espaços de endereçamento de memória leves, mecanismos para compartilhar regiões de memória e alta performance para *device drivers* a nível de usuário)[5] ele se mostra um microkernel da família L4 bastante indicado para estes casos.

A figura 2 mostra o funcionamento de um hypervisor em um ambiente como o fornecido pelo seL4. As VMMs interagem com o hypervisor para prover, gerir e controlar as VMs que atuam, no ponto de vista do usuário, como um sistema operacional onde podem executar aplicações e serem feitas interações. No ponto de vista do hypervisor ele gerencia através da comunicação com as VMMs quais componentes de *hardware* as VMs só podem fazer acesso [9].

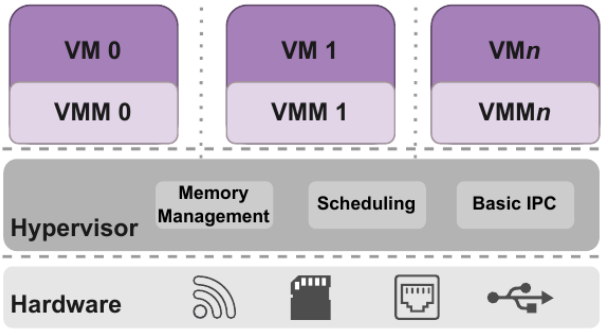

Figure 2: Forma com que um hypervisor do tipo 1 funciona [9].

A virtualização no seL4 é advinda de duas bibliotecas *libsel4vm*, uma biblioteca de virtualização de hardware para VMs para x86 e ARM, e *libsel4vmmplatsupport*, que contém diversos utilitários para VMMs e drivers para construção de VMs [14]. Além disso existem *frameworks*, como o CAmkES (*component architecture for microkernel-based embedded systems*), que utilizam a estrutura de virtualização seL4 para tornar mais simples a virtualização para o usuário.

3.0.2 CAmkES

O *Component Architecture for Microkernel-based Embedded Systems* (CAmkES) é uma arquitetura de componentes desenvolvida para sistemas embarcados baseados em microkernel, como o seL4. O CAmkES foi projetado para suportar a construção de sistemas seguros e de baixo overhead, proporcionando uma estrutura modular e extensível para o desenvolvimento de componentes de sistema e aplicativos.

A arquitetura do CAmkES é composta por várias camadas, incluindo componentes básicos, interfaces padrão, definições de componentes, implementações de componentes, serviços padrão e suporte a diversos padrões arquiteturais adequados para sistemas embarcados [7]. A base da arquitetura é o *core runtime*, que fornece o ambiente de execução e os serviços básicos necessários para o deploy de componentes CAmkES. Este runtime é projetado para suportar componentes e composições estáticas, o que significa que as instâncias de componentes são criadas apenas na inicialização do sistema, e as conexões entre componentes são estabelecidas no tempo de design, não podendo ser criadas ou modificadas dinamicamente em tempo de execução.

O modelo de componentes do CAmkES inclui vários elementos arquiteturais, como componentes, interfaces, conectores, conexões, composições e configurações. Um componente é a unidade básica de comportamento encapsulado e é usado para organizar operações e dados em interfaces que têm semânticas e comportamentos bem definidos. Os componentes podem ser passivos ou ativos, onde um componente passivo é semelhante a um objeto de nível de linguagem e não possui um thread de controle, enquanto um componente ativo contém seu próprio thread de controle. E esses componentes podem ser conectados para criar um componente composto (que é como uma aplicação completa é montada usando o CAmkES). A figura 3 exemplifica diversos componentes (elementos quadrados) conectados e suas interações.

As interfaces no CAmkES podem ser de três tipos: *remote procedure call* (RPC), eventos e dataports. As interfaces RPC definem a comunicação síncrona entre com-

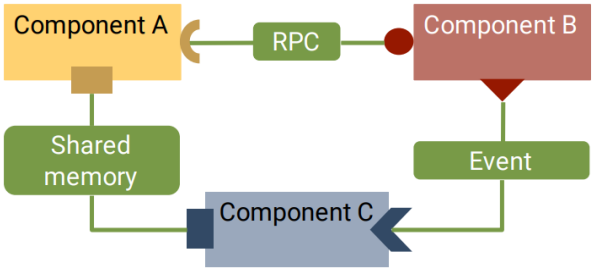

Figure 3: Aplicação construída com diversos elementos do CAmkES.

ponentes através de chamadas de procedimento remoto. Eventos são usados para notificações assíncronas entre componentes, e as interfaces de dataports representam variáveis compartilhadas que permitem a transferência de dados entre componentes.

4 Arquitetura da solução

A proposta envolve a modificação das primitivas de comunicação do seL4 para integrar algoritmos de criptografia pós-quântica utilizando a biblioteca *liboqs* [11], que fornece uma implementação robusta dos algoritmos padronizados pelo NIST [10]. Esta solução busca garantir a segurança da comunicação entre processos contra possíveis processos maliciosos que tentem adulterar mensagens ou realizar *eavesdropping*.

Para integrar a criptografia pós-quântica, se propôs a introdução de duas novas primitivas de comunicação no seL4: CryptoSend e CryptoRecv. A primitiva CryptoSend será responsável por adicionar uma camada de segurança às mensagens enviadas, realizando a geração de assinatura digital e a criptografia da mensagem. Antes de enviar uma mensagem, a primitiva CryptoSend utilizará o algoritmo Crystals-Dilithium para assinar digitalmente a mensagem, garantindo sua autenticidade e integridade. O processo de assinatura segue o Algoritmo 2, gerando uma assinatura σ que será anexada à mensagem. Após a assinatura, a mensagem será cifrada utilizando o algoritmo Crystals-Kyber. Este processo envolve a geração de uma chave simétrica compartilhada e o encapsulamento da mensagem cifrada (Algoritmo 4). A mensagem cifrada, juntamente com a assinatura, será então enviada ao destinatário.

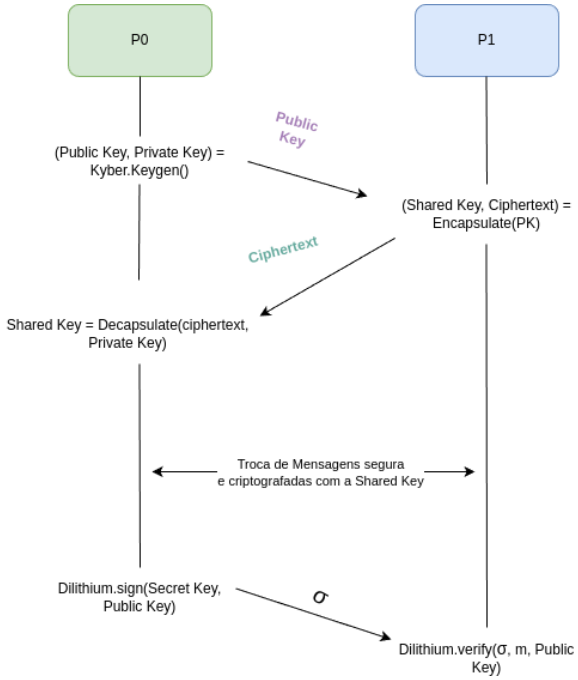

Figure 4: Estabelecimento de comunicação e assinatura das mensagens.

A primitiva CryptoRecv será responsável por decifrar e verificar a autenticidade das mensagens recebidas. Ao receber uma mensagem cifrada, a primitiva CryptoRecv utilizará o algoritmo Crystals-Kyber para desencapsular a mensagem e recuperar o texto original (Algoritmo 5). Após a descriptografia, a mensagem será verificada utilizando o algoritmo Crystals-Dilithium. A verificação da assinatura segue o Algoritmo 3, garantindo que a mensagem não foi alterada e que realmente foi enviada pelo remetente.

Para garantir que as novas primitivas sejam eficientes e seguras, a integração com o seL4 deve seguir os princípios de design do microkernel. Isso inclui respeitar o modelo de controle de acesso baseado em capacidades do seL4. Cada processo deve possuir as capacidades necessárias para utilizar as novas primitivas de comunicação. Além disso, a gestão de chaves criptográficas deve ser realizada de forma segura. A geração de chaves pode ser realizada durante o boot do sistema, garantindo que cada processo

possua uma chave pública e privada para as operações de criptografia e assinatura. A implementação das primitivas deve ser otimizada para minimizar a sobrecarga no desempenho do sistema, incluindo otimizações nas operações de criptografia e assinatura para garantir que a comunicação IPC continue a ser rápida e eficiente.

Ou seja, toda chamada para as novas primitivas CryptoSend ou CryptoRecv já garantiriam uma comunicação sendo criptografada e assinada com algoritmos pós-quânticos.

Essa seria uma aproximação de um cenário real onde algum processo, ou uma VM, tivesse um *passtrough* pelo microkernel e se comunicasse diretamente com um HW externo. Fazendo assim com que as garantias fornecidas pela verificação formal do microkernel do seL4 não fossem suficientes para garantir sua segurança e isolamento.

Integrar algoritmos de criptografia pós-quântica em um microkernel como o seL4 apresenta vários desafios. A complexidade da implementação é um dos principais desafios, devido à complexidade dos algoritmos e a necessidade de garantir que eles não introduzam vulnerabilidades. A adição de criptografia e verificação de assinatura pode introduzir latência nas operações de IPC, por isso é importante otimizar essas operações para minimizar o impacto no desempenho do sistema.

A modificação das primitivas de comunicação do seL4 para integrar algoritmos de criptografia pós-quântica é um passo importante para garantir a seguraça no cenário de computação quântica. Embora desafiadora, essa abordagem oferece uma solução robusta contra ameaças emergentes da computação quântica, garantindo autenticidade e a confidencialidade.

5 Conclusão

Este trabalho apresentou uma modificação nas primitivas de comunicação do microkernel seL4 para incorporar algoritmos de criptografia pós-quântica à nivel de IPC, incorporados da biblioteca *libqos*. Através da adição das primitivas CryptoSend e CryptoRecv, que integram os algoritmos Crystals-Kyber e Crystals-Dilithium, a comunicação entre processos contém uma camada adicional de segurança contra possíveis ataques de man-in-the-middle, falsificação de mensagens e também de eavesdropping advindas computadores quânticos. O *overhead* de comunicação acaba sendo significativo, dadas as quantidades de operações realizadas por cada algoritmo para garantir a segurança neste cenário, então ao se tomar a decisão de fazer uso do microkernel com estas novas primitivas acaba-se por aceitar o *trade-off* entre segurança e desempenho, o que dependendo da aplicação que fará uso pode fazer sentido. No mais, as primitivas de comunicação foram mantidas, então pode-se por optar por utilizar a camada de segurança apenas em momentos onde esse seja um fator crucial para a aplicação.

Para trabalhos futuros existem muitas direções que podem vir a ser exploradas. A realização de melhorias de performance através da integração de aceleradores criptográficos ao projeto faria com o que o *overhead* gerado pelo processo fosse menos perceptível.

A implementação da criptogtafia a nível de comunicação entre VMMs, quando o seL4 é utilizado como um hypervisor, faria com que se garantisse – principalmente quando uma VMM precisa fazer *passtrough* pelo seL4 para se comunicar com um *hardware* externo – a segurança a nível de comunicação externa também e não só entre processos.

Há também espaço para o desenvolvimento de provas formais para garantir que as modifcações que foram propostas conseguem manter as propriedades de segurança e também de corretude do seL4.

References

[1] Joppe Bos et al. “CRYSTALS - Kyber: A CCA-Secure Module-Lattice-Based KEM”. In: *2018 IEEE European Symposium on Security and Privacy (EuroSP)*. 2018, pp. 353–367. DOI: 10.1109/EuroSP.2018.00032.

[2] Lily Chen et al. *Report on post-quantum cryptography*. Vol. 12. US Department of Commerce, National Institute of Standards and Technology . . ., 2016.

[3] Léo Ducas et al. “Crystals–dilithium: Digital signatures from module lattices”. In: (2018).

[4] Léo Ducas et al. “CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018.1 (Feb. 2018), pp. 238–268.

DOI: 10.13154/tches.v2018.i1.238-268. URL: https://tches.iacr.org/index.php/TCHES/article/view/839.

[5] Gernot Heiser. "The role of virtualization in embedded systems". In: *Proceedings of the 1st Workshop on Isolation and Integration in Embedded Systems*. IIES '08. Glasgow, Scotland: Association for Computing Machinery, 2008, pp. 11–16. ISBN: 9781605581262. DOI: 10.1145/1435458.1435461. URL: https://doi.org/10.1145/1435458.1435461.

[6] Gerwin Klein et al. "seL4: formal verification of an OS kernel". In: *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*. SOSP '09. Big Sky, Montana, USA: Association for Computing Machinery, 2009, pp. 207–220. ISBN: 9781605587523. DOI: 10.1145/1629575.1629596. URL: https://doi.org/10.1145/1629575.1629596.

[7] Ihor Kuz et al. "CAmkES: A component model for secure microkernel-based embedded systems". In: *Journal of Systems and Software* 80.5 (2007). Component-Based Software Engineering of Trustworthy Embedded Systems, pp. 687–699. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2006.08.039. URL: https://www.sciencedirect.com/science/article/pii/S016412120600224X.

[8] Everton de Matos et al. "Integrating VirtIO and QEMU on seL4 for Enhanced Devices Virtualization Support". In: *2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. 2023, pp. 1076–1085. DOI: 10.1109/TrustCom60117.2023.00149.

[9] Everton de Matos and Markku Ahvenjärvi. "seL4 Microkernel for Virtualization Use-Cases: Potential Directions towards a Standard VMM". In: *Electronics* 11.24 (2022). ISSN: 2079-9292. DOI: 10.3390/electronics11244201. URL: https://www.mdpi.com/2079-9292/11/24/4201.

[10] National Institute of Standards and Technology (NIST). *NIST Post-Quantum Cryptography Standardization*. https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022. Accessed: 2024-06-13. 2022.

[11] Open Quantum Safe Project. *liboqs*. https://openquantumsafe.org/liboqs/. Accessed: 2024-06-30. 2024.

[12] Filip Opiłka et al. "Performance Analysis of Post-Quantum Cryptography Algorithms for Digital Signature". In: *Applied Sciences* 14.12 (2024). ISSN: 2076-3417. DOI: 10.3390/app14124994. URL: https://www.mdpi.com/2076-3417/14/12/4994.

[13] seL4 Project. *Frequently Asked Questions*. https://docs.sel4.systems/projects/sel4/frequently-asked-questions.html. Accessed: 2024-06-21. 2024.

[14] seL4 Project. *Virtualization on seL4*. https://docs.sel4.systems/projects/virtualization/. Accessed: 2024-06-25. 2024.

[15] Peter W Shor. "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer". In: *SIAM review* 41.2 (1999), pp. 303–332.

[16] Wikimedia Commons contributors. *OS structure*. https://upload.wikimedia.org/wikipedia/commons/thumb/6/67/OS-structure.svg/750px-OS-structure.svg.png?20190614091047. Accessed: 2024-06-18. 2019.