Vazamentos de Temporização e Exaustão de Entropia Quântica em ML-KEM (Kyber) com QRNG

A. C. P. Paixão, E. Mobilon, L. R. Pizzini, J. B. Rosolem, L. G. M. Riveros, M. Inga, T. Sutili, A. M. Braga e R. C. Figueiredo

CPQD – Centro de Pesquisa e Desenvolvimento em Telecomunicações
Campinas – SP – Brasil

{aportilho, mobilon, lpizzini, rosolem, lriveros}@cpqd.com.br,
{marvyn, tsutili, ambraga, rafaelcf}@cpqd.com.br

Abstract. *This paper investigates vulnerabilities in an ML-KEM (Kyber) key exchange environment integrated with a hardware Quantum Random Number Generator (QRNG). Our analysis found flaws on two fronts: in software, where the use of non-constant time functions (*`memcmp`*) caused timing side-channel leaks; and in the network, which enabled replay attacks. Although they did not break the cryptography, these attacks exhausted the system's quantum entropy, resulting in a Denial of Service (DoS). We mitigated these vulnerabilities by applying constant-time verification in the cryptographic core and Nonces in the application layer. Fuzzing tests confirmed the stability of the protected system. Finally, we discuss the adoption of the ETSI GS QKD 014 standard to overcome encapsulation latency.*

Resumo. *Este artigo investiga vulnerabilidades em um ambiente de troca de chaves ML-KEM (Kyber) integrado a um gerador quântico de números aleatórios (quantum random number generator – QRNG) em hardware. Nossa análise encontrou falhas em duas frentes: no software, onde o uso de funções não constantes (*`memcmp`*) gerou vazamentos por canal lateral de temporização; e na rede, que permitiu ataques de repetição. Embora não quebrassem a criptografia, esses ataques exauriram a entropia quântica do sistema, resultando em Negação de Serviço (denial of service – DoS). Mitigamos essas vulnerabilidades aplicando verificações em tempo constante no núcleo criptográfico e Nonces na camada de aplicação. Testes de fuzzing confirmaram a estabilidade do sistema protegido. Por fim, discutimos a adoção do padrão ETSI GS QKD 014 para contornar a latência do encapsulamento.*

1. Introdução

O avanço da computação quântica e a viabilidade de implementação do algoritmo de Shor [Shor 1994], que ataca os mecanismos clássicos assimétricos de troca de chaves e assinaturas digitais, levaram o NIST (*National Institute of Standards and Technology*) a padronizar o algoritmo de criptografia pós-quântica (*post-quantum cryptography* – PQC) CRYSTALS-Kyber (ML-KEM) [NIST 2024, NIST 2023], cuja segurança apoia-se na intratabilidade do problema M-LWE (*Module-Learning With Errors*) [Bos et al. 2022]. No entanto, a robustez matemática de um algoritmo não o torna imune a falhas de engenharia na sua implementação e operação.

Na prática, o sistema continua exposto em duas frentes, vulnerabilidades de *software* (como vazamentos por canal lateral de temporização) e riscos sistêmicos na camada de transporte (como ataques de repetição e negação de serviço). Embora a literatura recente foque em ataques físicos contra o Kyber em *hardware* embarcado [Ji and Dubrova 2023, Iavich and Kuchukhidze 2024], o impacto das falhas de rede sobre a fonte de entropia do sistema ainda é pouco investigado experimentalmente.

Para preencher essa lacuna, desenvolvemos um *testbed* integrando o ML-KEM a uma rede TCP/IP. Como diferencial, substituímos o gerador pseudoaleatório padrão do sistema operacional por um módulo gerador quântico de números aleatórios (QRNG) com interface USB. Nosso principal objetivo é avaliar como essas vulnerabilidades se manifestam e, sobretudo, como ataques de rede podem ser usados para exaurir os recursos físicos do *hardware* quântico. Nas seções a seguir, detalhamos a arquitetura do experimento, a injeção do QRNG, os ataques realizados e as defesas arquiteturais propostas.

2. Trabalhos Relacionados

A segurança do experimento depende de duas camadas — código e rede. No nível do código, embora o Kyber seja matematicamente forte, ele pode sofrer vazamentos por canal lateral de temporização [Ji and Dubrova 2023, Iavich and Kuchukhidze 2024]. O uso de funções padrão em C, como o `memcmp`, interrompe comparações no primeiro erro (*early exit*), criando diferenças de tempo que revelam falhas da chave ao atacante [Kocher 1996]. Para evitar isso, o padrão exige funções de tempo constante, como a `verify` [Bos et al. 2022].

Além das vulnerabilidades diretas de implementação e dos ataques de repetição, a literatura ressalta a importância da resiliência do sistema contra o esgotamento de entropia e entradas malformadas. Implementações criptográficas puramente em software tipicamente utilizam geradores pseudoaleatórios (pseudo-random number generator - PRNG) baseados em diretrizes como as do [NIST 2015], que podem comprometer a segurança caso o estado interno (seed) seja exposto. Ao integrar um QRNG em *hardware* [Turan et al. 2018], nosso cenário elimina a dependência de algoritmos determinísticos para a geração de chaves. Contudo, isso introduz o desafio de proteger essa fonte física de entropia pura contra sobrecargas de rede. Por isso, diferentemente de avaliações puramente teóricas, complementamos nossa análise com testes de *fuzzing* [OWASP 2024], demonstrando que uma sanitização rigorosa na camada de aplicação é essencial para manter a estabilidade do servidor sob condições extremas de tráfego.

Na camada de rede, ataques sistêmicos são perigosos mesmo sob criptografia forte. Em um ataque de repetição [Syverson 1994], reenvios de pacotes antigos não revelam a chave, mas forçam o servidor a refazer cálculos pesados, causando negação de serviço (DoS) [Stallings 2020].

Nosso trabalho preenche uma lacuna prática na literatura ao avaliar como esses ataques exaurem não apenas a CPU, mas também os recursos físicos de um *hardware* quântico.

3. Metodologia

Montamos um ambiente Linux (Intel Core i7) simulando um cliente (Bob), um servidor (Alice) e um atacante (Eve). Assumindo a matemática do Kyber (ML-KEM-768

[NIST 2023]) como inquebrável, o objetivo do atacante é derrubar a comunicação injetando anomalias [OWASP 2024] ou repetindo pacotes [Menezes et al. 1996]. Para mitigar isso, dividimos a arquitetura: a rede opera em Python, agindo como um escudo ativo que descarta pacotes maliciosos na aplicação. Já o núcleo criptográfico em C foca em velocidade e substitui a função `randombytes` nativa para extrair aleatoriedade pura de um módulo QRNG com interface USB (Crypta Labs QCicada) [Crypta Labs 2025], operando em modo contínuo e eliminando o uso de sequências de bits pseudoaleatórias (pseudo-random bit sequence – PRBS) em software [NIST 2015, Turan et al. 2018]

O primeiro passo metodológico testou o vazamento de temporização no código. Compilamos duas versões do desencapsulamento sem otimizações automáticas (GCC `-O0`): a versão segura, usando `verify` em tempo constante [Bos et al. 2022], e a intencionalmente vulnerável, usando `memcmp`, que encerra a leitura no primeiro erro (*early exit*) [Kocher 1996]. Medimos o tempo de 50.000 iterações, tanto remotamente (caixa-preta) [Kocher 1996], quanto diretamente nos ciclos do processador em C (caixa-branca).

A etapa final avaliou a sobrevivência do sistema sob estresse de rede. Simulamos ataques de repetição reenviando pacotes antigos [Syverson 1994]. Como defesa, implementamos *Nonces* (16 *bytes*) efêmeros no Python [NIST 2015]. Requisições sem o *Nonce* exato eram derrubadas imediatamente, poupando a CPU e o nosso gerador quântico. Em seguida, testamos interceptações (*man-in-the-middle* – MitM) [Stallings 2020], bloqueadas no cliente via ancoragem de *hash* da chave [Menezes et al. 1996]. Por fim, bombardeamos a porta do servidor com pacotes malformados (*fuzzing*) para atestar a estabilidade [OWASP 2024] e concluímos medindo o tempo total de 100 *handshakes* sequenciais para atestar a viabilidade [NIST 2023].

4. Resultados

Nesta seção, apresentamos os resultados dos testes de ataques descritos, avaliando desde o vazamento no processador até a resistência do tráfego de rede.

4.1. Avaliação de Vazamento por Canal Lateral de Temporização

No cenário de caixa-preta, o ruído natural das operações matemáticas pesadas do Kyber, como a transformação NTT e o *hashing* SHAKE [Bos et al. 2022], mascarou completamente o vazamento de temporização. Textos válidos e inválidos levaram quase o mesmo tempo de processamento remoto (entre 67.000 e 69.000 ns). Isso significa que o sistema parece seguro.

Porém, ao medirmos diretamente os ciclos do processador (caixa-branca) ao redor da função de comparação, a falha estrutural ficou evidente. A Tabela 1 mostra que a implementação segura manteve um tempo constante impecável. Já na versão vulnerável, o uso do `memcmp` fez com que a rejeição de pacotes inválidos fosse 30,5% mais rápida [Kocher 1996].

Tabela 1. Resultados do Teste de Vazamento de Temporização Focada na Implementação Vulnerável.

Cenário	**Tempo médio (ns)**	**Desvio padrão**
Texto cifrado válido (varredura completa)	45,95	Médio
Texto cifrado inválido (saída antecipada)	19,92	Médio
Diferença	-26,03 ns	∼56,6%

Essa assimetria de mais de ∼25 nanossegundos não é um ruído ocasional. Ela cria uma assinatura temporal estatisticamente clara, como mostrado na Figura 1, permitindo que um invasor local deduza o conteúdo da chave.

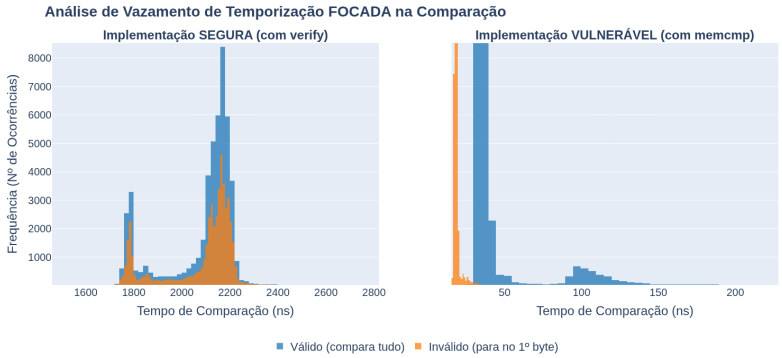

Figura 1. Comparação de temporizações: à esquerda (segura), há sobreposição total; à direita (vulnerável), a separação clara confirma o vazamento de temporização.

4.2. Defesas de Rede, Robustez e Desempenho

Na camada de transporte, demonstramos o impacto de não haver uma validação de sessão. Como mostra a Figura 2, ao injetarmos um pacote antigo, o servidor aceitou o processamento. Embora a chave não tenha sido quebrada devido à efemeridade do algoritmo, o servidor desperdiçou tempo de CPU e consumiu entropia valiosa do módulo QRNG inutilmente, configurando uma severa negação de serviço (DoS).

Ao adicionar o *Nonce* na camada de aplicação, o cenário mudou. A Figura 3 mostra que o *script* em Python detectou imediatamente a ausência do *Nonce* atualizado e derrubou a conexão. Isso blindou o núcleo em C, preservando a entropia quântica de forma integral. De forma similar, a nossa defesa por ancoragem de chave (*key pinning*) bloqueou instantaneamente uma tentativa de interceptação ativa (MitM) na porta 8082, assim que o cliente notou a divergência de *hash* (Figura 4). Para atestar que o sistema não cederia a falhas de *software*, bombardeamos a porta do servidor com o *fuzzing*. Pacotes vazios, cabeçalhos malformados e cargas gigantes (10 MB) foram injetados e o servidor lidou com todas as anomalias de forma controlada, mantendo o serviço ativo (Figura 5).

Por fim, medimos a viabilidade de rodar toda essa infraestrutura no mundo real. Em um lote de 100 *handshakes* sequenciais (Figura 6), o sistema processou tudo com 100% de sucesso em apenas 2,49 segundos. Atingir a marca de quase 40 conexões por segundo (com tempo médio inferior a 25 ms por troca), considerando a latência do Python e a matemática densa do Kyber, atesta que o ML-KEM-768 é plenamente viável para operação contínua.

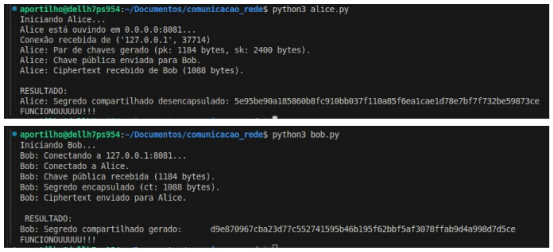

Figura 2. Servidor processa pacote antigo do atacante e gera um segredo inválido.

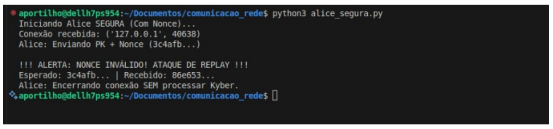

Figura 3. Servidor detecta divergência no *Nonce* e encerra a conexão, poupando recursos.

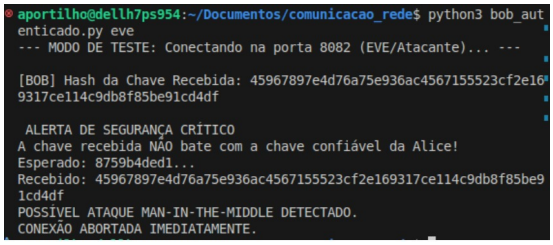

Figura 4. Defesa MitM: cliente detecta chave falsa via *hash* e aborta a conexão.

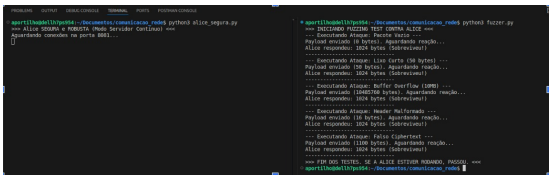

Figura 5. Teste de *fuzzing*: servidor sobrevive aos vetores e permanece online.

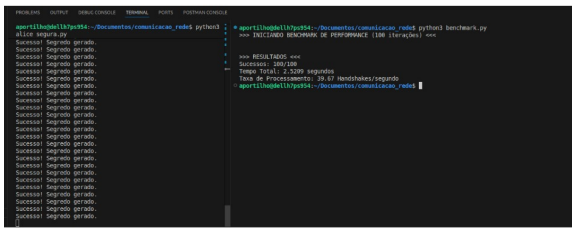

Figura 6. Log do teste de desempenho com resultado de ∼39,67 conexões/segundo.

5. Conclusão

A transição pós-quântica exige não apenas matemática robusta, mas engenharia segura. Nossos resultados mostraram que a segurança do Kyber depende fortemente da implementação e do protocolo de rede.

No código, o uso de funções não constantes (`memcmp`) reintroduziu vulnerabilidades clássicas [Kocher 1996]. Observamos que otimizações de compilador (GCC `-O3`) podem mascarar parcialmente essas falhas no binário, reforçando a importância da auditoria prática e do uso de funções em tempo constante, como a `verify` [Bos et al. 2022].

Na rede, evidenciamos que ataques de repetição [Syverson 1994] exauriam a entropia do QRNG [Turan et al. 2018, NIST 2015]. Nosso escudo de *Nonces* em Python mitigou esse risco bloqueando conexões maliciosas. Somado à resistência contra *fuzzing* [OWASP 2024] e ao desempenho viável [NIST 2023], atestamos que a divisão arquitetural proposta (rede em Python e criptografia em C) protege com eficácia tanto os dados quanto os recursos físicos de hardware.

6. Trabalhos Futuros

A principal limitação deste experimento é o gargalo da geração de chaves sob demanda (em tempo real) durante o *handshake*. Para solucionar isso, o nosso próximo passo é evoluir essa arquitetura para um sistema de gerenciamento de chaves (*key management system* - KMS) híbrido.

Nesse novo modelo, o servidor utilizará o QRNG para gerar chaves de forma contínua, as quais serão então distribuídas via Kyber e armazenadas em bancos de dados (*buffers* assíncronos). Quando Alice e Bob precisarem se comunicar, eles solicitarão chaves ao KMS através do protocolo ETSI GS QKD 014. Um mecanismo baseado na transmissão de identificadores leves (*key IDs*) assegura o sincronismo entre as chaves simétricas resgatadas em ambos os lados do enlace de comunicação. Espera-se que essa abordagem mitigue problemas de latência.

Referências

Bos, J. W., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J. M., Schwabe, P., Seiler, G., and Stehlé, D. (2022). CRYSTALS-Kyber: Algorithm specifications and supporting documentation.

Crypta Labs (2025). *QCicada QRNG User Guide*. London. Version 1.6.

Iavich, M. and Kuchukhidze, T. (2024). Investigating CRYSTALS-Kyber vulnerabilities: Attack analysis and mitigation. *Cryptography*, 8(2):15.

Ji, Y. and Dubrova, E. (2023). A side-channel attack on a masked hardware implementation of CRYSTALS-Kyber. In *Proceedings of the 2023 Workshop on Attacks and Solutions in Hardware Security (ASHES '23)*. ACM.

Kocher, P. C. (1996). Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology – CRYPTO '96*, pages 104–113. Springer.

Menezes, A. J., Van Oorschot, P. C., and Vanstone, S. A. (1996). *Handbook of Applied Cryptography*. CRC Press.

NIST (2015). Recommendation for random number generation using deterministic random bit generators. Technical Report SP 800-90A Rev.1, NIST.

NIST (2023). FIPS 203 (draft): Module-Lattice-Based Key-Encapsulation Mechanism Standard. Technical report, NIST, Gaithersburg, MD.

NIST (2024). Post-Quantum Cryptography Standardization.

OWASP (2024). Fuzzing.

Shor, P. W. (1994). Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134. IEEE.

Stallings, W. (2020). *Cryptography and Network Security: Principles and Practice*. Pearson, 8 edition.

Syverson, P. F. (1994). A taxonomy of replay attacks. In *Proceedings of the 7th Computer Security Foundations Workshop*. IEEE Computer Society Press.

Turan, M. S. et al. (2018). Recommendation for the entropy sources used for random bit generation. Technical Report SP 800-90B, NIST, Gaithersburg, MD.