Post Quantum Hard-to-Find Bugs

STEINBACH Matteo University of Luxembourg
Email: matteo.steinbach.001@student.uni.lu
This report has been produced under the supervision of:
Peter ROENNE University of Luxembourg
Email: peter.roenne@uni.lu
Johann GROZSCHÄDL University of Luxembourg
Email: johann.groszschaedl@uni.lu

***Abstract*—This is a scientific research report made in the context of a bachelor semester project BSPS6. It is made by STEINBACH Matteo, a 3rd year student of the Bachelor in Computer Science in Luxembourg under the supervision of Dr. ROENNE Peter and Prof. GROßSCHÄDL Johann**

The report expands the research made in the previous semester on elusive bugs in cryptographic implementations, by looking at post quantum cryptography. The advent of quantum computing poses an existential threat to classical cryptographic systems, necessitating the adoption of post-quantum cryptography (PQC). However, the novelty of PQC algorithms introduce unique challenges for secure implementations particularly without hard-to-find bugs (HFBs) that evade conventional testing. This report investigates HFBs in PQC schemes specifically focusing on those with lattice-based foundations. Following the research work and found HFBs we provide a suite of Known Answer Tests (KATs) tailored to PQC primitives.

The research will be continued in the summer and be transformed in a complete paper.

I. Introduction

The advent of quantum computing poses a significant and existential threat to the foundations of classical cryptographic systems. Algorithms such as RSA and Elliptic Curve Cryptography (ECC), which underpin much of contemporary digital security, are rendered vulnerable to quantum algorithms like Shor's algorithm, capable of solving integer factorization and discrete logarithm problems in polynomial time. In response to this impending cryptographic paradigm shift, the National Institute of Standards and Technology (NIST) has initiated a standardization process for post-quantum cryptography (PQC) schemes, including lattice-based algorithms like Kyber and Dilithium, hash-based SPHINCS+, and Falcon's compact lattice signatures. However, the transition to PQC introduces unique and complex challenges for secure implementations. Unlike classical cryptography, which has undergone decades of cryptanalysis and refinement, PQC algorithms are relatively novel, relying on intricate mathematical structures such as polynomial rings, lattice problems, and hash forests that are distinct from their classical counterparts. This novelty, coupled with the relative immaturity of PQC codebases and a scarcity of domain-specific testing tools, significantly increases the risk of subtle, hard-to-find bugs (HFBs) that can critically undermine cryptographic guarantees and evade conventional testing methodologies.

Hard-to-find bugs (HFBs) are elusive implementation errors that manifest under rare, specific conditions, making their detection exceptionally challenging due to their low trigger probability, algorithmic subtlety, and cross-layer complexity. While HFBs in classical cryptography often stem from issues like carry propagation or modular reduction errors, PQC implementations face distinct challenges related to polynomial coefficient management, Number-Theoretic Transform (NTT)-domain arithmetic, and compliance with error distribution specifications. For example, a single missed coefficient reduction in Kyber's polynomial multiplication can corrupt ciphertexts, and improper Gaussian sampling in Dilithium may inadvertently leak secret key bits through timing side channels. The elusive nature of HFBs makes them prime targets for adversaries, who can craft precise inputs or operational sequences to exploit them, potentially leading to severe consequences such as cryptographic failures, unauthorized data access, or private key leakage. Moreover, there is an increasing concern regarding maliciously introduced HFBs, especially in large-scale open-source projects and new implementations.

This report, which extends prior research on HFBs in classical cryptography, addresses these new challenges in the post-quantum context. Specifically, this study aims to answer the following scientific questions:

- How do the unique mathematical foundations and computational characteristics of PQC schemes lead to new categories of hard-to-find bugs compared to classical cryptography?
- What systematic methodologies can be developed to identify, classify, and document these PQC-specific implementation vulnerabilities?
- How can existing classical testing methodologies be adapted and new tools be developed to effectively detect and mitigate elusive vulnerabilities in PQC implementations, thereby ensuring their readiness for quantum-safe deployment?

In addressing these questions, this report makes several notable contributions to the field of secure PQC implementation:

- Systematic characterization of PQC-specific HFBs: A classification system is developed and applied to PQC schemes, identifying new patterns of vulnerabilities distinct from classical cryptography, such as those related to polynomial coefficient management, NTT-domain arith-

metic, and error distribution compliance. Over 10 vulnerabilities were identified and documented, particularly in lattice-based cryptographic implementations.
- Detailed analysis of lattice-based vulnerabilities: The report provides an in-depth analysis of common software-based attack techniques targeting standardized lattice-based PQC schemes, including side-channel attacks on NTT operations (e.g., timing leaks, DVFS frequency leaks, CPA on intermediates), fault injection attacks (e.g., PRNG state corruption, Keccak stuck-at faults), power analysis on discrete Gaussian samplers (e.g., rejection sampling leakage, sign bit leakage), and plaintext-checking oracle attacks. Specific vulnerabilities and their implications for Falcon, Dilithium, and Kyber implementations are highlighted.
- Comprehensive countermeasure strategies: Unified mitigation strategies are presented for various vulnerability domains, including constant-time arithmetic, blinded memory access, leakage-resistant sampling, computational redundancy, state verification, error-correcting codes, constant-time verification, response uniformization, ciphertext sanitization, compiler-level protections, and unified masking strategies.
- Development of a tailored Known Answer Test (KAT) suite: A suite of Known Answer Tests (KATs) is developed, specifically tailored to PQC primitives and NIST PQC finalists like Falcon and Kyber. This involves extending Google's Wycheproof framework to support PQC schemes, including the development of JSON schema definitions and a strategy for generating test vectors.
- Demonstration of bug reproduction and input crafting: The report details the process of reproducing known implementation bugs, such as Falcon's acceptance of mutated signatures due to floating-point instabilities, and expanding the test vector suite through input crafting and mutation (e.g., bit-flipping, adding redundant zero-padding, introducing single-byte offsets) to expose edge-case vulnerabilities and detect over-acceptance behavior.

These contributions collectively provide a roadmap for securing PQC implementations against elusive vulnerabilities, ensuring their readiness for quantum-safe deployment and contributing to the robustness of future cryptographic infrastructures.

II. Hard-to-Find Bugs (HFBs)

Hard-to-Find Bugs (HFBs) in cryptographic software are elusive implementation errors that manifest under rare, specific conditions. Their detection is challenging due to their low trigger probability, algorithmic subtlety, and cross-layer complexity. Despite their rarity, HFBs can critically undermine cryptographic guarantees, especially in public key cryptography (PKC).

A. Characteristics

HFBs are defined by:
- *Rarity*: They are triggered only under uncommon input combinations, execution sequences, or environmental conditions.
- *Subtlety*: Arise from intricate interactions in arithmetic, memory, or randomness generation.
- *Low observability*: Traditional testing fails to surface them without targeted inputs or symbolic analysis.
- *Cross-disciplinarity*: Require insights from cryptography, systems, hardware, and programming languages for mitigation.

The criteria was used in a loser way, as there are way less bugs including PQC since the schemes are so new.

B. Impact and Exploitability

1) Impact: The impact of bugs is assessed by the severity of the vulnerabilities they introduce, enabling developers to prioritize fixes based on potential damage. Key factors for evaluating impact include:

- Exposure of sensitive data
- System integrity compromise
- Exploitability by attackers
- Long-term implications for system security

This classification is crucial for understanding cryptographic risks and underscores the need for expert knowledge in developing secure systems.

HFBs pose a significant security risk due to their potential exploitability by attackers. Their elusive nature makes them prime targets for adversaries who can craft precise inputs or operation sequences to exploit them. This can lead to severe consequences, such as cryptographic failures, unauthorized data access, or even private key leakage, compromising the integrity and confidentiality of secure systems.

2) Exploitability: Given their low probability of occurrence under normal conditions, such bugs can remain undetected for extended periods. This is especially important for the near transition to new systems that have not yet passed the test of time and are under heavy cryptanalysis.

Moreover, there is an increasing risk of maliciously introduced HFBs, particularly in environments with frequent and large-scale code contributions, such as open-source projects. This risk is especially significant for malicious nation-state actors particularly with pqc as they will most likely be the ones with quantum computer capacities, who may seek to embed hidden backdoors in software widely regarded as secure, such as cryptographic libraries, to enable long-term espionage or control over critical systems. Attackers could intentionally introduce subtle HFBs during major commits or feature additions, which may evade detection during standard reviews. A recent example is the 2024 XZ Utils incident, where a malicious actor used social engineering to gain trust and insert a subtle backdoor into the software, potentially allowing unauthorized code execution on Debian machines [**?**]. These flaws are often masked within complex or large code changes, especially in low-level languages like C, C++, and Assembly, which are commonly used in cryptography implementations for performance reasons. Assembly, in particular, is still prevalent for critical optimizations, adding to the challenge of detecting such bugs.

III. Initial Research on Hard-to-Find Bugs in Classical Cryptography

At the end of the BSPS5 program, the initial research on hard-to-find bugs (HFBs) in classical public-key cryptography was largely completed. This foundational work focused on identifying and analyzing subtle implementation bugs in classical asymmetric cryptographic algorithms, particularly RSA and Elliptic Curve Cryptography (ECC). These systems depend on intricate mathematical operations such as long-integer modular arithmetic, which are especially prone to subtle and security-critical implementation errors. The primary objective was to understand, classify, and develop detection methodologies for these bugs that can undermine cryptographic security.

A. Key Initial Contributions and Findings

The research addressed the identification, classification, and mitigation of HFBs in classical cryptographic implementations. Key contributions included:

- Systematic collection and classification of over 50 real-world HFBs, mainly resulting from low-level implementation errors. The following categories were identified:
  - Carry propagation errors, accounting for approximately 30% of cases, arising when intermediate carries in multi-precision arithmetic operations are not correctly propagated across word boundaries.
  - Mismanagement of cryptographic state or context, such as incorrect initialization or propagation of internal states and flags.
  - Incorrect implementation of cryptographic algorithms, involving mistakes in arithmetic logic or failure to follow formal specifications.
  - Missing input validation, leading to issues like buffer overflows or crashes when malformed or incorrect inputs are not rejected.
  - Misconfiguration or inconsistent parameter handling, especially regarding primes, generators, or initialization vectors.
  - Failure to ensure constant-time operations, allowing timing side-channel attacks based on secret-dependent execution times.
- Evaluation of applicable testing methodologies:
  - Differential testing across implementations to identify inconsistencies.
  - Static analysis, including formal methods like Cryptol, to check for correctness and timing issues.
  - Monte Carlo testing using random inputs to simulate diverse real-world conditions.
  - Fuzzing with adversarial inputs to reveal edge-case failures.
  - Known Answer Tests (KATs) comparing outputs to reference values.
- Proposal of a standardized testing framework combining differential testing, KATs, and continuous fuzzing, with an emphasis on integrating formal verification for high-assurance cryptographic software.
- Development of a practical C-based Known Answer Test suite derived from Google's Wycheproof, adapted for libraries such as OpenSSL, targeting the detection of known HFBs.

B. From Research to Publication

After completing the initial research which expanded a little into the second semester and implementation, the work was refined into a research paper to submit. This involved learning about academic writing structures, submission platforms such as EasyChair, and editing/shortening the content to fit a 22-page format. The paper was submitted to the ARES conference, a venue with low acceptance rate. Despite rejection, the reviews—ranging from critical to encouraging—provided constructive feedback and noted inexperience with academic writing.

This experience was a valuable learning process for me and gave very interesting feedback. Based on the reviews, a plan was made to revise and extend the paper by integrating the post-quantum cryptography (PQC) components of the project. The new version is intended for submission to the Sec-itc conference in September.

IV. Background on Post-Quantum Cryptography

Post-quantum cryptography (PQC) addresses the existential threat quantum computers pose to classical public-key cryptography. Shor's algorithm solves integer factorization and discrete logarithm problems in polynomial time [53], compromising RSA and ECC security. Grover's algorithm provides quadratic speedups against symmetric primitives [34], reducing effective key strength. PQC leverages mathematical problems with conjectured quantum resistance, necessitating a fundamental cryptographic transition. This section details major PQC families, standardization progress, and implementation hurdles.

A. Fundamental Categories of Post-Quantum Cryptography

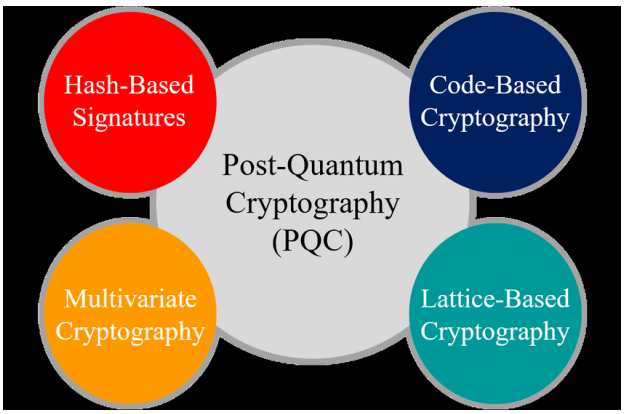

Fig. 1: PQC categories

PQC algorithms are classified into five categories based on underlying mathematical structures [14]. Each offers distinct security-efficiency tradeoffs:

1) Lattice-Based Cryptography: The most mature PQC family leverages NP-hard problems like Learning With Errors (LWE) [51]. Security derives from shortest vector problem (SVP) hardness in high-dimensional lattices, with classical reductions from worst-case GapSVP to LWE [11]. The core LWE problem requires recovering secret $\mathbf{s} \in \mathbb{Z}_q^n$ from samples $(\mathbf{a}, b \approx \langle \mathbf{a}, \mathbf{s} \rangle \mod q)$.

Tradeoffs: Strong security reductions but larger keys than classical schemes (Kyber-768: 1,188 bytes). Signature sizes vary significantly (Dilithium: 2,420 bytes; Falcon: 666 bytes), values from FIPS.

Notable schemes:

- Kyber (ML-KEM): Module-LWE KEM [10]
- Dilithium (ML-DSA): Module-LWE signature [19]
- Falcon: NTRU-based signature [23]
- FrodoKEM: Unstructured LWE KEM [9]

2) Code-Based Cryptography: Based on NP-hardness of decoding random linear codes [4]. McEliece (1978) with Goppa codes remains quantum-resistant, relying on syndrome decoding complexity.

Tradeoffs: Longest security history but large keys (Classic McEliece: 1MB+). Information Set Decoding remains best attack [3].

Notable schemes:

- Classic McEliece [42]
- HQC [24]

3) Hash-Based Cryptography: Security reduces to collision resistance of hash functions [13]. Grover's algorithm provides only quadratic speedup [5].

Tradeoffs: Minimal assumptions but large signatures (SPHINCS+-128s: 8,080 bytes) [6].

Notable schemes:

- SPHINCS+ (SLH-DSA): Stateless signature [7]
- LMS/XMSS: Stateful schemes [22], [43]

4) Multivariate Polynomial Cryptography: Security depends on MQ-problem hardness: solving systems of quadratic equations over finite fields. Signature schemes like Rainbow reached NIST Round 3 [18] but suffered key recovery attacks [8].

Tradeoffs: Fast verification but parameter sensitivity [**?**].

Current status: Active research on attack-resistant variants [56].

5) Isogeny-Based Cryptography: Based on hardness of computing isogenies between supersingular elliptic curves [17]. SIKE's 2022 break demonstrated subexponential attacks [15].

Tradeoffs: Previously offered smallest keys (SIKEp434: 330 bytes) but severely compromised security [25].

B. NIST Standardization Process

NIST initiated PQC standardization in 2016 with 82 submissions [49]. After four evaluation rounds:

- FIPS 203 [46]: Kyber (KEM)
- FIPS 204 [45]: Dilithium (signatures)
- FIPS 205 [47]: SPHINCS+ (signatures)
- FIPS 206 [44]: Falcon (signatures)
- HQC

Kyber/Dilithium were selected for efficiency-security balance [2].

C. International Standardization

Global bodies are developing complementary frameworks:
ISO/IEC JTC 1:

- Standardizing Classic McEliece (ISO 18033-4) [30]
- Draft standard for FrodoKEM [31]
- Developing quantum-safe cryptography standards [29]

Other Bodies:

- ETSI: Hybrid key exchange (TS 104 015) QKD [21]
- IETF: TLS/SSH integration (PQUIP WG) [28]
- ANSSI: Hybrid deployment guidelines [1]

D. Implementation Challenges

Key adoption barriers:

- Performance Overheads:
  - Kyber encapsulation: 1.5× slower than ECDH [32]
  - SPHINCS+ verification: 100× more cycles than EdDSA [33]
- Protocol Integration:
  - TLS handshake expansion beyond 1500B MTU [50]
  - X.509 certificate chain bloat (Dilithium: +15KB) [57]
- Hardware Constraints:
  - Falcon signature generation requires floating-point units
  - McEliece key storage exceeds IoT device limits [54]
- Cryptographic Agility: Coexistence of classical/PQC algorithms increases implementation complexity/efficiency.

Hybrid approaches (e.g., ECDH + Kyber or with qkd) provide transitional security and allow the scheme to be secure as long as one of the scheme still stays they in practice make attacks way harder and provide security against most attackers even if a quantum computer exists. These are ellaborated upon in the appendix A.

V. Diverse Mathematical Foundations

Classical cryptosystems like RSA and ECC are based on number-theoretic problems such as integer factorization and elliptic curve discrete logarithms (big modulo arithmetic). In contrast, post-quantum cryptosystems rely on different mathematical foundations, including lattice-based problems, code-based cryptography, hash based and isogeny-based cryptography (polynomial arithmetic) precise more

A. Lattice-Based Cryptography

Lattice-based cryptography derives security from the computational hardness of problems in structured lattices. We begin with the formal mathematical definition of lattices and associated hard problems before specializing to polynomial rings and practical schemes.

1) Mathematical Foundations of Lattices: A lattice $\mathcal{L}$ is a discrete additive subgroup of $\mathbb{R}^m$ spanned by a basis. Formally, for n linearly independent vectors $B = \{b_1, \ldots, b_n\} \subset \mathbb{R}^m$, the lattice $\mathcal{L}$ is:

$$\mathcal{L}(B) = \left\{ \sum_{i=1}^{n} \alpha_i b_i \mid \alpha_i \in \mathbb{Z} \right\}.$$

The rank is n and dimension is m. When $n = m$, $\mathcal{L}$ is a full-rank lattice. The fundamental parallelepiped is $\mathcal{P}(B) = \{\sum_{i=1}^{n} x_i b_i \mid 0 \le x_i < 1\}$.

The dual lattice $\mathcal{L}^*$ is defined as:

$$\mathcal{L}^* = \{y \in \text{span}(\mathcal{L}) \mid \forall v \in \mathcal{L}, \langle y, v \rangle \in \mathbb{Z}\}.$$

Key lattice invariants include:

- Minimum distance: $\lambda_1(\mathcal{L}) = \min_{v \in \mathcal{L} \setminus \{0\}} \|v\|$
- Successive minima: $\lambda_i(\mathcal{L}) = \inf\{r \mid \dim(\text{span}(\mathcal{L} \cap \overline{B}(0, r))) \ge i\}$
- Covering radius: $\mu(\mathcal{L}) = \sup_x \min_{v \in \mathcal{L}} \|x - v\|$

2) Hard Computational Problems: Security relies on the hardness of these problems for arbitrary lattices:

- Shortest Vector Problem (SVP): Given basis B, find $v \in \mathcal{L}(B) \setminus \{0\}$ minimizing $\|v\|$.
- Shortest Independent Vectors Problem (SIVP): Find n linearly independent vectors $v_1, \ldots, v_n \in \mathcal{L}$ where $\max \|v_i\| \le \lambda_n(\mathcal{L})$.
- Bounded Distance Decoding (BDD): Given $t \in \mathbb{R}^m$ with $\text{dist}(t, \mathcal{L}) < \gamma \cdot \lambda_1(\mathcal{L})$ for $\gamma > 0$, find the closest lattice vector.

These problems are NP-hard for exact solutions and remain hard for polynomial approximation factors under randomized reductions. [12]

3) Ideal Lattices and Algebraic Structure: Practical schemes use ideal lattices derived from polynomial rings. Let $R = \mathbb{Z}[x]/(f(x))$ where $f(x)$ is monic and irreducible. For $f(x) = x^n + 1$, this ring is well-defined only when n is a power of 2; otherwise, $f(x)$ may not be irreducible. An ideal $I \subset R$ generates a lattice via coefficient embedding $\sigma : R \to \mathbb{Z}^n$:

$$\sigma\left(\sum_{i=0}^{n-1} a_i x^i\right) = (a_0, a_1, \ldots, a_{n-1})^\intercal.$$

The ideal lattice is $\mathcal{L}_I = \sigma(I)$. For $f(x) = x^n + 1$ with $n = 2^k$, this yields a cyclotomic lattice with geometric structure enabling efficient operations.

4) Ring Learning With Errors (RLWE) Problem: Fix ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$. While $q \equiv 1 \pmod{2n}$ ensures the existence of a primitive $2n$-th root of unity, practical schemes (e.g., Kyber with $q = 3329$) use incomplete-NTT or negacyclic convolution when this condition is unmet. [39]

Let χ be an error distribution (typically discrete Gaussian). The decision-RLWE problem is to distinguish between:

- Samples $(a_i, b_i = a_i \cdot s + e_i)$ where $a_i \leftarrow R_q$, $s \leftarrow \chi$, $e_i \leftarrow \chi$
- Uniform samples $(a_i, u_i) \leftarrow R_q \times R_q$

The search-RLWE problem is to recover s from samples. RLWE reduces to worst-case ideal lattice problems (e.g., quantum reduction from approx-SVP).

5) Polynomial Arithmetic in R_q: Set $R_q = \mathbb{Z}_q[x]/(x^n + 1)$. Polynomial multiplication uses negacyclic convolution:

$$c(x) = a(x) \cdot b(x) \mod (x^n + 1, q)$$

Coefficient reduction modulo q is straightforward. Polynomial reduction modulo $x^n + 1$ exploits the identity $x^n \equiv -1$, enabling recursive computation. For $a(x) = \sum_{i=0}^{n-1} a_i x^i$ and $b(x) = \sum_{j=0}^{n-1} b_j x^j$:

$$c_k = \sum_{\substack{i+j=k \\ 0 \le i,j < n}} a_i b_j - \sum_{\substack{i+j=n+k \\ 0 \le i,j < n}} a_i b_j$$

6) Number Theoretic Transform (NTT): The NTT arises from the ring isomorphism:

$$R_q \cong \prod_{i=0}^{n-1} \mathbb{Z}_q[x]/(x - \zeta^{2i+1}) \cong \mathbb{Z}_q^n,$$

where ζ is a primitive $2n$-th root of unity modulo q (if it exists). The NTT maps $a(x)$ to $(a(\zeta^1), a(\zeta^3), \ldots, a(\zeta^{2n-1}))$.

The NTT is evaluation at roots of $x^n + 1$:

$$\text{NTT}(a) = (\hat{a}_0, \ldots, \hat{a}_{n-1}) \quad \text{where} \quad \hat{a}_i = a(\zeta^{2i+1})$$

Multiplication in the NTT domain is coefficient-wise:

$$\text{NTT}(c) = \text{NTT}(a) \odot \text{NTT}(b) \implies \hat{c}_i = \hat{a}_i \cdot \hat{b}_i \mod q$$

Complexity is $O(n \log n)$ using Cooley-Tukey butterfly operations. Decryption correctness requires $\|e\|_\infty < q/4$ for Kyber and $\|e\|_\infty < q/2$ for Dilithium due to different norm bounds.

7) Error Analysis and Decryption: Consider ciphertext $c = a \cdot s + e \in R_q$. Decryption computes:

$$m' = c - a \cdot s' = e + a \cdot (s - s')$$

For correctness, we require $m' = e$ when $s = s'$ and $\|e\|_\infty < q/2$. The error term must satisfy:

$$|e_i| < \frac{q}{2} \quad \text{for all coefficients } i$$

to avoid modular wrapping. In practice, coefficients of e follow a discrete Gaussian distribution $D_{\mathbb{Z},\sigma}$ with $\sigma \approx 3.2$ for Kyber-1024. Failure probability is bounded via:

$$\Pr[\|e\|_\infty \ge q/2] \le 2n \cdot \exp\left(-\frac{(q/2)^2}{2\sigma^2}\right)$$

B. Code-Based Cryptography

1) Algebraic Coding Theory: Let $\mathcal{C}$ be a linear $[n, k, d]_q$ code over $\mathbb{F}_q$ with minimum distance d satisfying the Gilbert-Varshamov bound:

$$q^{n-k} \ge \sum_{i=0}^{d-2} \binom{n}{i} (q-1)^i$$

The weight enumerator $A(z) = \sum_{w=0}^{n} A_w z^w$ satisfies the MacWilliams identity:

$$A^{\perp}(z) = q^{-k}(1+(q-1)z)^n A\left(\frac{1-z}{1+(q-1)z}\right)$$

For t-error correction, we require $d \geq 2t+1$. The covering radius ρ is the minimal r such that $\bigcup_{c\in\mathcal{C}} B(c,r) = \mathbb{F}_q^n$ where $B(c,r)$ is the Hamming ball of radius r. [40]

2) Syndrome Decoding Complexity: The syndrome decoding problem (SDP) is: given $H \in \mathbb{F}_q^{(n-k)\times n}$, $s \in \mathbb{F}_q^{n-k}$, find $e \in \mathbb{F}_q^n$ with $(e) \leq t$ such that $He^\top = s$. This is NP-complete for binary codes. The information set decoding (ISD) complexity is:

$$C_{\text{ISD}} = \min_{0\leq w\leq t} \binom{k}{p}^{-1}\binom{n}{t}^{-1}\binom{n-k}{t-w}\binom{k}{w}(q-1)^w$$

where p is the number of errors in the information set. For binary Goppa codes, the best attack is Stern's algorithm with complexity:

$$O\left(\binom{n}{t}/\binom{n-k}{t}\right)$$

3) Goppa Code Construction: Let $g(x) \in \mathbb{F}_{2^m}[x]$ be monic irreducible of degree t, and $L = \{\alpha_1,\ldots,\alpha_n\} \subset \mathbb{F}_{2^m}$ with $g(\alpha_j) \neq 0$. The Goppa code $\Gamma(L,g)$ has parity-check matrix:

$$H = \begin{bmatrix} g(\alpha_1)^{-1} & g(\alpha_2)^{-1} & \cdots & g(\alpha_n)^{-1} \\ \alpha_1 g(\alpha_1)^{-1} & \alpha_2 g(\alpha_2)^{-1} & \cdots & \alpha_n g(\alpha_n)^{-1} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{t-1} g(\alpha_1)^{-1} & \alpha_2^{t-1} g(\alpha_2)^{-1} & \cdots & \alpha_n^{t-1} g(\alpha_n)^{-1} \end{bmatrix} \cdot V$$

where V is the Vandermonde matrix $V_{ij} = \alpha_j^{i-1}$. The minimum distance $d \geq 2t+1$. Patterson's decoder solves the key equation:

$$\Gamma(x)S(x) \equiv \omega(x) \mod g(x)$$

where $\Gamma(x) = \prod(x-\alpha_i)$, $S(x)$ is the syndrome polynomial, and $\deg\omega < \deg\Gamma$.

4) McEliece Cryptosystem: Let G_{sys} be systematic generator matrix for $\Gamma(L,g)$. The public key is $\tilde{G} = SG_{\text{sys}}P$ where:

- $S \in \text{GL}_k(\mathbb{F}_2)$ random invertible
- $P \in \text{Sym}(n)$ random permutation

Encryption: $c = m\tilde{G} + e$ with $(e) = t$. Decryption:

1) Compute $cP^{-1} = mSG_{\text{sys}} + eP^{-1}$ Apply Patterson to decode mS Recover $m = (mS)S^{-1}$

The work factor for structural attacks is $O(2^{mt})$ for classical and $O(2^{mt/2})$ for quantum.

C. Hash-Based Cryptography

1) Winternitz One-Time Signature: For message digest $d \in \{0,1\}^n$, split into s blocks of $\log_2 w$ bits. Parameters:

$$\ell_1 = \left\lceil \frac{n}{\log_2 w} \right\rceil, \quad \ell_2 = \left\lfloor \frac{\log_2(\ell_1(w-1))}{\log_2 w} \right\rfloor + 1, \quad \ell = \ell_1 + \ell_2$$

Secret key: $sk = (x_1,\ldots,x_\ell) \in_R (\{0,1\}^n)^\ell$. Public key: $pk = (H^{w-1}(x_1),\ldots,H^{w-1}(x_\ell))$. Signature for d:

1) Compute $b_i = \sum_{j=1}^{\ell_1} d_{(i-1)\log_2 w+j}\cdot 2^{j-1}$ for $1 \leq i \leq \ell_1$ Compute checksum $C = \sum_{i=1}^{\ell_1}(w-1-b_i)$ Split C into ℓ_2 blocks b_i $(\ell_1 < i \leq \ell)$ Output $\sigma_i = H^{b_i}(x_i)$

Verification requires $H^{w-1-b_i}(\sigma_i) = pk_i$. Security reduces to second-preimage resistance with advantage $\text{Adv}_{\text{WOTS}}^{\text{EU-CMA}} \leq \ell \cdot \text{Adv}_H^{\text{SPR}}$. [52]

2) Merkle Tree Construction: Let T_h be a binary tree of height h with 2^h leaves. Define:

$$\begin{aligned} \text{Leaf: } \ell_i &= H(\text{OTS.pk}_i) \\ \text{Internal: } N_{u,v} &= H(N_{u-1,2v} \parallel N_{u-1,2v+1}) \\ \text{Root: pk} &= N_{h,0} \end{aligned}$$

The authentication path for leaf i is the sequence of siblings $(a_0,\ldots,a_{h-1})$ where a_u is the sibling of node on path from ℓ_i to root. Signature consists of:

$$\sigma = (\text{OTS.sig}, \text{OTS.pk}, (a_0,\ldots,a_{h-1}), \text{index})$$

Verification reconstructs path to root. The security level is $\min\{2^{n/2}, 2^h\}$ against collision attacks.

3) SPHINCS+ Construction: SPHINCS+ uses multiple layers:

- Hyper-tree of height h with d layers, each having k-ary subtrees
- Few-time signature (FTS) FORS: t trees of k leaves, signs $\log_2(tk)$ bits
- Pseudorandom function PRF : $\{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$

Key generation:

$$(\text{sk}, \text{pk}) = (\text{SK}, H(\text{root}_d \parallel \text{root}_{\text{FORS}}))$$

Signing uses address randomization: $\text{adr}_{\text{node}} = \text{PRF}(\text{SK}, \text{msg} \parallel \text{index})$. Security level λ requires $n \geq 2\lambda$ and $h \geq \lambda/\log_2 d$.

D. Multivariate Polynomial Cryptography

1) MQ Problem Complexity: The multivariate quadratic (MQ) problem: given m polynomials $P^{(k)}(x_1,\ldots,x_n) = \sum_{i\leq j}\alpha_{ij}^{(k)}x_ix_j + \sum_i \beta_i^{(k)}x_i + \gamma^{(k)}$ over $\mathbb{F}_q$, find $x \in \mathbb{F}_q^n$ satisfying all equations. This is NP-complete for exact solving. The HFE (Hidden Field Equations) system uses the univariate map:

$$Q(y) = \sum_{i=0}^{d}\sum_{j=0}^{i} c_{ij}y^{q^i+q^j}$$

with $d = O(\log q^n)$. The MinRank problem: given matrices $M_1,\ldots,M_k \in \mathbb{F}_q^{m\times n}$, find λ_i such that $\text{rank}(\sum\lambda_i M_i) \leq r$. This is NP-hard for $r \geq 2$. [37]

2) Oil-Vinegar Trapdoor: In the unbalanced Oil-Vinegar (UOV) scheme [41]:

- Vinegar variables: $v_1, \ldots, v_v \in \mathbb{F}_q$
- Oil variables: $o_1, \ldots, o_o \in \mathbb{F}_q$
- Central map: $q_k(\mathbf{v}, \mathbf{o}) = \sum_{i=1}^{v}\sum_{j=1}^{o} \alpha_{ij}^{(k)} v_i o_j + \sum_{i=1}^{v}\sum_{j=i}^{v} \beta_{ij}^{(k)} v_i v_j + \sum_{i=1}^{v} \gamma_i^{(k)} v_i + \sum_{j=1}^{o} \delta_j^{(k)} o_j + \eta^{(k)}$

The ratio v/o is typically 2-3. To invert:

1) Assign random values to vinegar variables
2) Solve linear system in oil variables
3) If singular, repeat from step 1

The public map is $F = S \circ Q \circ T$ with $S \in \mathrm{GL}_m(\mathbb{F}_q)$, $T \in \mathrm{GL}_n(\mathbb{F}_q)$. The direct attack complexity is $O(q^v \cdot o^3)$.

E. Isogeny-Based Cryptography

1) Supersingular Isogeny Graphs: Let $p = \ell_A^{e_A} \ell_B^{e_B} f \pm 1$ be prime. The supersingular isogeny graph $\mathcal{G}_\ell(\mathbb{F}_{p^2})$ has:

- Vertices: j-invariants of supersingular curves
- Edges: isogenies of degree ℓ
- Diameter: $\lceil \log_\ell p \rceil$
- Ramanujan property: $\lambda_2 \leq 2\sqrt{\ell - 1}/\ell$

The isogeny $\phi : E \to E/\langle P \rangle$ is computed via:

$$\phi(x, y) = \left(N(x)/D(x)^2, yN'(x)/D(x)^3\right)$$

where $N(x) = \prod_{Q \in \langle P \rangle \setminus \{0\}} (x - x_Q)$ and $D(x) = \prod_{Q \in \langle P \rangle \setminus \{0\}} (x - x_Q)^{1/2}$.

F. Conclusion

Post-quantum cryptographic schemes fundamentally differ from classical systems in their security foundations, mathematical structures, and operational characteristics. The principal distinctions are categorized below.

1) Security Assumptions: Classical cryptography relies on number-theoretic problems:

- Integer factorization: Given $N = pq$, recover p, q (RSA)
- Discrete logarithm: Given $g^x \mod p$, find x (Diffie-Hellman, DSA)
- Elliptic curve discrete logarithm: Given $[k]P$, find k (ECDH, ECDSA)

These problems are vulnerable to Shor's quantum algorithm, which solves them in polynomial time $O((\log n)^3)$. Post-quantum alternatives depend on problems resistant to known quantum attacks:

- Lattice: Shortest Vector Problem (GapSVP_γ), Learning With Errors (LWE)
- Coding: Syndrome Decoding Problem (NP-complete)
- Multivariate: MQ problem (NP-hard over finite fields)
- Isogeny: Supersingular Isogeny Path Finding

The best known quantum attacks against these require subexponential or exponential time.

Dimension	Classical	Post-Quantum
Algebraic structure	Cyclic groups Finite fields	Module lattices Polynomial rings R_q Code spaces $\mathcal{C} \subset \mathbb{F}_q^n$ Isogeny graphs
Key derivation	Modular exponentiation Point multiplication	NTT convolution Syndrome computation Isogeny evaluation
Hardness reduction	Random oracle model Generic group model	Worst-case to average-case (lattice-based)

VI. Bugs and Potential for Attacks

This study focuses on lattice-based cryptographic schemes due to their computational efficiency and prominent position in post-quantum standardization efforts **??**. While the primary analysis concentrates on lattice constructions, a comprehensive bug classification system is maintained to accommodate vulnerabilities found across all post-quantum cryptographic paradigms.

A. Methodology for Vulnerability Discovery and Classification

A systematic approach was developed for identifying and analyzing implementation vulnerabilities in post-quantum cryptography:

1) Source Investigation: Comprehensive examination of:
   - Academic publications and conference proceedings
   - Reported CVEs and vulnerability databases
   - PQC standardization discussion forums (e.g., NIST PQC Google Group)
   - Issue trackers for reference implementations (PQClean, SUPERCOP, OpenSSL3.5, AWS, LibOQS, IBM's lattice-crypto and more)
2) Triage Process: Each potential vulnerability was evaluated against the criteria defined in Section II, with particular attention to:
   - Mathematical soundness of the underlying vulnerability
   - Practical exploitability in real-world deployments
   - Impact on security guarantees
3) Documentation Framework: Validated vulnerabilities were recorded using the following structured format:
   a) `Specification`: Cryptographic context and affected components
   b) `Defect`: Precise characterization of the implementation flaw
   c) `Impact`: Security consequences and violation of cryptographic properties
   d) `Code Snippet`: Representative code exhibiting the vulnerability

1) Cross-Paradigm Vulnerability Taxonomy: The documented vulnerabilities were classified according to the following taxonomy, which applies to public-key cryptography implementations generally but reveals particular patterns in post-quantum contexts:

- `CARRY_PROPAGATION.md`: Mismanagement of carry chains in multi-precision arithmetic
- `CRYPTO_STATE.md`: Improper handling of secret-dependent state transitions

- `IMPLEMENTATIONS.md`: Deviations from mathematical specifications
- `INPUT_VALIDATION.md`: Inadequate handling of adversarial inputs
- `PARAM_HANDLING.md`: Flaws in parameter encoding/decoding
- `CONSTANT_TIME.md`: Violations of constant-time execution requirements

Notably, the distribution of vulnerability types differs significantly between classical and post-quantum implementations. Carry propagation issues, which constituted approximately 30% of vulnerabilities in classical systems don't happen in post quantum cryptography.

Carry propagation is rare across post-quantum cryptographic schemes due to the absence of large multi-precision arithmetic. Lattice-based constructions operate over rings such as

$$R_q = (\mathbb{Z}/q\mathbb{Z})[X]/(X^n + 1)$$

with bounded coefficients and small modulus q, avoiding limb-based carries. Code-based schemes (e.g., McEliece) rely on linear algebra over $\mathbb{F}_2$, while multivariate and hash-based schemes use operations over small finite fields or bitwise hash functions. These designs inherently avoid long carry chains typical in RSA and ECC.

Through this methodology, 10+ vulnerabilities were identified and documented in lattice-based cryptographic implementations. The following subsections analyze representative cases and their security implications, with particular focus on standardized lattice constructions due to their immediate practical relevance. Other post-quantum cryptography types will be addressed in future work.

B. Common Vulnerabilities in Lattice-Based Schemes

This subsection details software-based attack techniques targeting standardized lattice-based post-quantum cryptographic schemes, categorized by shared attack mechanisms.

1) Side-Channel Attacks Targeting Number-Theoretic Transforms: Lattice-based schemes frequently employ the Number-Theoretic Transform (NTT) for polynomial arithmetic, introducing multiple side-channel vulnerabilities:

- Butterfly Operation Leakage (Dilithium/Kyber): Radix-2 butterfly operations $(a, b) \mapsto (u, v)$ where $u = (a + \omega b) \bmod q$, $v = (a - \omega b) \bmod q$ exhibit three vulnerabilities:
  1) Intermediate Value CPA: Hamming weight $\mathrm{HW}(w)$ of pre-reduction product $w = a + \omega b$ correlates with power consumption. Known ω enables recovery of secret b (e.g., $\mathbf{s}_1$ coefficients) from $\omega b \bmod q$ via CPA.
  2) Timing Leakage: Conditional subtraction $u \leftarrow u - q$ when $u \geq q$ introduces data-dependent cycle delays Δt. Observed latency leaks the most significant bit (MSB) of ωb, satisfying $\lfloor \omega b/q \rfloor = 0$ for $u < q$.
  3) DVFS-Based Frequency Leakage (Kyber): Vectorized butterflies exhibit power consumption $P_{\text{butterfly}} = P_0 + \delta(\mathrm{HW}(x) + \mathrm{HW}(y))$. CPU frequency scaling $f_{\text{CPU}} \approx f_{\max} - \kappa P$ distinguishes zero/non-zero coefficients. Crafted decapsulation inputs $\widehat{u}'_j = 1$ reveal support of $\widehat{\mathbf{s}}$ via f_{CPU} fluctuations during INTT. [38]

2) Fault Injection Attacks: Deliberate perturbation of computation enables key recovery through differential analysis:

- PRNG Faults in Deterministic Signing:
  - *FALCON*: Single-bit fault in ChaCha20 state alters randomness $\rho \to \rho'$. Signature pairs $\sigma = (s_1, s_2)$, $\sigma' = (s'_1, s'_2)$ satisfy $f \star (s_2 - s'_2) \equiv (s_1 - s'_1) \pmod q$. Short vectors $(\delta s_2, \delta s_1)$ form NTRU lattice vectors recoverable via Babai/LLL reduction. [35]
  - *Dilithium*: Fault-induced ρ' modifies $y \to y'$. Signature equations yield $A(z' - z) \equiv 0 \pmod q$ with short $\Delta z = y' - y$. Lattice reduction on $\Lambda = \{(u, v) : Au - v \equiv 0 \bmod q\}$ recovers secrets. [48]
- Keccak-f State Perturbation (Dilithium): Stuck-at fault during θ-step at (x^*, y^*, z^*) modifies SHAKE256 output. Altered challenge c' produces signature $\mathbf{z}' = \mathbf{y} + c'\mathbf{s}_1$. Differencing with valid signature gives $\mathbf{z}' - \mathbf{z} = (c' - c)\mathbf{s}_1 + (\mathbf{y}' - \mathbf{y})$. Sparse $c' - c$ leaks linear equations in $\mathbf{s}_1$, solvable via τ-sparse linear algebra. [55]

3) Power Analysis on Discrete Gaussian Samplers: Gaussian sampling routines leak secret-dependent information through power consumption:

- Rejection Sampling Leakage: `BaseSampler` loop iterations for $k \leftarrow U(0, K)$ expose magnitude through rejection probability $\exp(-k^2/(2\sigma^2))$. Observed loop count distribution $\Pr[L > \ell | k] = (1 - \exp(-k^2/(2\sigma^2))^\ell$ reveals k via covariance analysis $\mathrm{Cov}(P_i, \mathbf{1}\{k_i = k'\})$. [20]
- Sign Bit Leakage: `SamplerZ` sign flip operation $z = (-1)^b |z|$ leaks b via power traces. Hamming distance $\mathrm{HD} = |\mathrm{HW}(\neg m) - \mathrm{HW}(m)|$ for $b = 1$ (vs. 0 for $b = 0$) enables threshold-based recovery.
- Single-Trace Negation Leakage (FALCON Key Generation): Negation of shifted secret $w = \lfloor u/2^{63} \rfloor$ leaks via Hamming weight difference $\Delta = |\mathrm{HW}(-u) - \mathrm{HW}(u)|$. Distinguishes $u = 0$ ($\Delta = 0$) from $u = -1$ ($\Delta = 1$) per coefficient. [16]

4) Plaintext-Checking Oracle Attacks: When Dilithium is used in hybrid encryption or IBE, an attacker may build a plaintext-checking oracle from verification timing. Verification checks

$$\mathbf{z} \leq B, \quad \|A\,\mathbf{z} - \mathbf{h} \star \mathbf{t}\|_\infty \leq \eta,$$

aborting early if the second condition fails. Define

$$\mathrm{PC}_{\mathsf{sk}}(m) = \begin{cases} 1 & \text{if signature on } m \text{ passes all checks,} \\ 0 & \text{otherwise.} \end{cases}$$

By submitting crafted pairs $(\mathbf{z}', \mathbf{h}')$ that encode guesses of secret bits and observing early-abort timing, the attacker learns bits of $\mathbf{s}_1$ in $O(n \log q)$ queries. Once $\mathbf{s}_1$ is known, $\mathbf{s}_2$ follows from public equations [36].

Kyber's CCA check recomputes

$$\mathbf{v}'' = \mathrm{Enc}(\mathbf{u}', \mathbf{s}), \quad \text{if } \texttt{poly\_compress}(\mathbf{v}'') \neq$$

`c.short` then reject.

If rejection timing varies when $\mathbf{v}'' \neq \mathbf{v}$, the attacker obtains

$$\mathrm{PC}_{\mathbf{s}}(\mathbf{v}, \mathbf{u}') = \begin{cases} 1, & \mathbf{v}'' = \mathbf{v}, \\ 0, & \text{otherwise.} \end{cases}$$

By flipping bits in $\mathbf{u}'$ or $\mathbf{v}$ and measuring early-abort timing, one performs a binary search on each coefficient in $O(\kappa \log q)$ queries, recovering $\mathbf{s}$ and breaking CCA security.

C. Falcon specific

Falcon relyes heavily on floating-point arithmetic for fast sampling and signature generation. However, its reliance on high-precision computations and deterministic randomness generation introduces several side-channel and fault-based vulnerabilities, here are presented possible common attacks against falcon implementations.

1) Floating-Point Mantissa Leakage: During FFT-based lattice sampling, Falcon processes complex polynomial coefficients of the form

$$\widehat{a}_j = u_j + i\, v_j, \quad u_j, v_j \in \mathbb{R},$$

where u_j and v_j are stored using IEEE-754 double-precision format with 52-bit mantissas $m_{u,j}$ and $m_{v,j}$. On AVX-512 platforms, simultaneous vector loads cause EM emissions

$$E(t) \propto \sum_j \left[\mathrm{HW}(m_{u,j}) + \mathrm{HW}(m_{v,j})\right],$$

where $\mathrm{HW}(\cdot)$ denotes the Hamming weight. Correlation Power Analysis (CPA) on traces $E_i(t)$ enables recovery of $\widehat{a}_j$ coefficients. Applying inverse FFT and lattice reduction then yields the secret trapdoor polynomials f and g.

2) Floating-Point Timing and Power Side Channels: The discrete Gaussian sampler in Falcon, `gauss_sampler()`, involves data-dependent operations such as division, trigonometric functions, and modular reductions. Let $x \sim D_\sigma$ be a sampled value. Internally, Falcon computes

$$r = \frac{x}{\sigma}, \quad u = \mathtt{fmod}(x, 2\pi), \quad p = \exp\left(-\frac{1}{2}r^2\right).$$

The computation time for $\mathtt{fmod}(x, 2\pi)$ scales with the mantissa m of x:

$$x = (-1)^s \cdot 2^e \cdot \left(1 + \frac{m}{2^{52}}\right), \quad m \in [0, 2^{52} - 1].$$

An attacker can measure timings T_i and perform differential analysis using the model

$$T_i \approx \alpha + \beta \cdot \mathrm{HW}(m_i) + \varepsilon_i,$$

with noise term ε_i. Recovered mantissas reveal bits of the sampled x_i, and since signature components depend on these samples, secret key recovery follows.

3) Randomness State Recovery: In deterministic mode, Falcon generates ephemeral randomness using a ChaCha20-based PRNG:

$$\mathsf{PRNG} : \{0,1\}^{256} \to \{0,1\}^{512}.$$

Each internal state S_t is updated via ARX operations:

$$S_{t+1} = (S_t + k_t) \lll r_t \oplus c_t,$$

where k_t is a subkey word, r_t a rotation, and c_t a constant. Power traces $P(t)$ satisfy the leakage model:

$$P(t) \approx \gamma + \delta \cdot \mathrm{HW}(S_t \oplus k_t) + \zeta_t.$$

By solving the minimization problem

$$\min_{\rho} \sum_{t=0}^{T-1} \left|P(t) - \mathrm{HW}(\rho_t \oplus k_t)\right|,$$

an attacker can recover the ephemeral output ρ. This enables deterministic prediction of future Gaussian samples and compromise of the private key.

4) Floating-Point Rounding Discrepancies: Different hardware implementations or compiler options may result in inconsistent evaluation of expressions like

$$r = \frac{x}{\sigma}, \quad r^2, \quad p = \exp\left(-\frac{1}{2}r^2\right),$$

particularly when using FMA instructions on x86 versus standard double-precision on ARM. If

$$r^2 \bmod 1 \in [1 - \varepsilon, 1), \quad \varepsilon = O(2^{-53}),$$

then rejection sampling decisions may differ by ± 1 on the final coefficients. Comparing two signatures σ_1, σ_2 under different rounding modes yields a difference vector:

$$\Delta z = z^{(1)} - z^{(2)} \in \{-1, 0, 1\}^n,$$

and the corresponding polynomial difference satisfies

$$\Delta s_1 = \mathtt{fft}^{-1}(h) \star \Delta s_2.$$

If Δs_2 is sparse (support size ≤ 2), then this forms a short linear convolution that reveals constraints on $\widehat{h}$, which can be solved over multiple samples to recover the secret key [27].

5) Fault Injection Attacks: Physical fault injection (e.g., via voltage glitching or clock manipulation) during FFT or sampling can induce single-bit faults in coefficients. Let the faulty output be σ', and define

$$e = \sigma - \sigma'.$$

Then $e(z)$ has low Hamming weight and satisfies

$$\mathtt{fft}(e) \approx 0.$$

Recovering e from this approximate null-space equation allows reconstruction of faulty polynomial differences, which can expose parts of the trapdoor basis. Countermeasures include redundant FFT validation and modular arithmetic checks.

D. Dilithium (ML-DSA)

Dilithium

1) Intermediate Value Leakage and Template Attacks: Let $A \in \mathbb{Z}_q^{n\times n}$ be the public matrix and let $\mathbf{s}_1, \mathbf{s}_2 \in \mathbb{Z}_q^n$ be the secret vectors. During signing, compute

$$\mathbf{w} = A\,\mathbf{s}_1 + \mathbf{s}_2 \mod q,$$

and split

$$\mathbf{w}_0 = \left\lfloor \frac{\mathbf{w}}{2^d} \right\rceil, \quad \mathbf{w}_1 = \mathbf{w} \bmod 2^d,$$

for some integer d. The sampler uses $\mathbf{w}_0$ to draw $\mathbf{y} \in \{-\kappa, \ldots, \kappa\}^n$, then forms $\mathbf{z} = \mathbf{y} + c\,\mathbf{s}_1$.

A profiling phase records power traces $P_k(t)$ for known values $w_{0,i} = j$ at branch points b, computing class means $\mu_j^{(b)}$ and covariances $\Sigma_j^{(b)}$. In the attack phase, for an observed trace $P^*(t)$, the log-likelihood for each class j is

$$\ell(j) = -\tfrac{1}{2}\big(P^* - \mu_j^{(b)}\big)^T \Sigma_j^{(b)^{-1}} \big(P^* - \mu_j^{(b)}\big),$$

and the estimated value is $\hat{\jmath} = \arg\max_j \ell(j)$. Recovering a sufficient fraction of $\{\hat{w}_{0,i}\}$ allows solving

$$\mathbf{w}_0\, 2^d + \mathbf{w}_1 = A\,\mathbf{s}_1 + \mathbf{s}_2 \mod q$$

via lattice reduction. Empirical results indicate $\sim 5\times10^3$ traces achieve $> 99\%$ accuracy and full key recovery in under 2^{50} operations [26].

E. Kyber (ML-KEM)

Kyber achieves CCA security via a Fujisaki–Okamoto transform on a CPA-secure PKE scheme over

$$R_q = (\mathbb{Z}/q\mathbb{Z})[X]/(X^n{+}1), \quad q = 3329,\ n = 256,\ k \in \{2,3,4\}.$$

Implementations must address compiler-level timing leaks, DVFS/NTT frequency side channels, masking/fault faults, and plaintext-checking oracles. This section analyzes core attacks and outlines mitigations.

1) Compiler Optimization Leaks (KyberSlash1): During decapsulation, `SharedSecret` computes

$$\mathbf{m}' = \mathbf{v} - \mathbf{u}\,\mathbf{s} \bmod q, \quad \mathbf{s} \in R_q^k,$$

then maps each bit via

$$\mu_i = \begin{cases} 0, & m_i = 0, \\ q-1, & m_i = 1, \end{cases} \quad \widehat{m}_i = \left\lfloor \tfrac{2\,m_i' + \lfloor q/2 \rfloor}{q} \right\rceil \bmod 2.$$

In C under `gcc -Os`, the division by KYBER_Q compiles to a variable-time `idiv`. Its cycle count satisfies

$$T_i \approx \alpha + \beta\,\lfloor \log_2(2\,m_i' + \lfloor q/2 \rfloor) \rfloor + \varepsilon_i.$$

By choosing ciphertexts that force m_i' into small or large ranges, and measuring T_i, an attacker recovers high-order bits of $\langle \mathbf{u}, \mathbf{s} \rangle_i \bmod q$. Selecting $\mathbf{u} = \mathbf{e}_j$ yields direct leakage of s_j, leading to full key recovery in $\sim 2{,}048$ decapsulations on x86-64 :contentReference[oaicite:0]index=0.

2) Ciphertext Compression Timing (KyberSlash2): After inverse NTT, decapsulation re-encrypts

$$\mathbf{v}'' = \mathbf{u}'\,\mathbf{s} + \boldsymbol{\epsilon}, \quad \mathbf{u}' = \mathrm{NTT}^{-1}(\widehat{\mathbf{u}}'),$$

then compresses each coefficient via

$$c_i'' = \left\lfloor \frac{2^d\, v_i'' + q/2}{q} \right\rceil \bmod 2^d, \quad d \in \{3,4\}.$$

The core C code again uses `idiv`, leaking

$$T_i \approx \alpha + \beta\,\lfloor \log_2(2^d\, v_i'' + \lfloor q/2 \rfloor) \rfloor + \varepsilon_i.$$

By fixing $\mathbf{u}' = \mathbf{e}_\ell$, one forces $v_\ell'' = s_j + \epsilon_\ell \bmod q$ and distinguishes whether s_j lies in the lower or upper half of $[0, q)$. Repeating for each index recovers all s_j in $O(n)$ decapsulations on ARM and x86 platforms :contentReference[oaicite:1]index=1.

3) Masked Decapsulation Faults: With first-order masking, each secret coefficient is split:

$$s_j = s_j^{(1)} + s_j^{(2)} \bmod q, \quad s_j^{(1)}, s_j^{(2)} \xleftarrow{\$} \mathbb{Z}_q.$$

A fault that sets $s_j^{(1)} = 0$ causes the de-masked value $\widetilde{s}_j = s_j^{(2)}$ to be uniform in $\mathbb{Z}_q$. This alters the distributions of $\widetilde{m}_i'$ or $\widetilde{v}_i''$ during decoding or compression. By measuring side-channel leakage and statistically distinguishing pre- and post-fault distributions at index j, an attacker identifies the faulted index. Repeating with fresh masks across $\approx 10^4$ decapsulations recovers all s_j with $> 90\%$ success on Cortex-M4 :contentReference[oaicite:2]index=2.

F. Countermeasures

This section presents unified mitigation strategies for vulnerabilities in lattice-based cryptographic implementations. The countermeasures are categorized by protection domain rather than specific schemes, with rigorous technical specifications applicable to Falcon, Dilithium, Kyber, and related constructions.

1) Side-Channel Protection:

1) Constant-Time Arithmetic:
   - Replace variable-time divisions with Montgomery reduction: $x_{\text{red}} = x \cdot R \mod q$ where $R = 2^k > q$
   - Implement Barrett reduction without branches: $z \leftarrow x - q \cdot \lfloor (x \cdot \mu) \gg k \rfloor + \mathsf{mask}(x - q)$
   - Use unconditional subtraction-addition: $u \leftarrow (a + \omega b) + q \cdot \mathbb{1}_{[0,2q)}(a + \omega b)$
2) Blinded Memory Access:
   - Apply arithmetic masking to polynomial coefficients: $\widetilde{a}_i = a_i + r_i \mod q$ with $r_i \sim U(0, q-1)$
   - Randomize NTT butterfly sequence: $\pi(\text{operations}) \leftarrow \mathsf{Perm}(n)$ per invocation
3) Leakage-Resistant Sampling:
   - Implement discrete Gaussian sampler with constant-time CDT: Precompute $T[\cdot]$ with fixed rejection iterations
   - Add timing noise: $t_{\text{sample}} \leftarrow t_{\text{actual}} + \lfloor \mathcal{N}(0, \sigma_t^2) \rfloor$ where $\sigma_t > 100$ cycles

2) Fault Injection Mitigation:

1) Computational Redundancy:
   - Dual-rail execution: Compute $y = f(x)$ and $y' = f(x)$ on isolated hardware units, verify $y \equiv y'$
   - Algebraic checksums: Embed $\sigma = \sum_{i=0}^{n-1} s_i \mod 2^{32}$ in key material
2) State Verification:
   - Keccak integrity checks: $\mathsf{CRC32}(S_i) \stackrel{?}{=} \mathsf{CRC32}(S_{i-1} \oplus \theta(S_{i-1}))$ after each round
   - PRNG state duplication: Maintain (ρ, ρ') with $\rho' = \rho \oplus \Delta$, verify Δ before output
3) Error-Correcting Codes:
   - Apply Reed-Solomon codes: Encode RCDT tables with $\mathrm{RS}(255, 223)$ correction capability
   - Use Hamming codes for coefficient storage: $s_i \mapsto (s_i \| \mathsf{Ham}(s_i))$

3) Oracle Attack Prevention:

1) Constant-Time Verification:
   - Always perform full norm checks: Compute $\|\mathbf{z}\|_\infty$ unconditionally
   - Implement fixed-latency comparison: $\delta \leftarrow \bigvee_{i=0}^{n-1}(c_i'' \oplus c_i)$ via bitwise OR
2) Response Uniformization:
   - Add random delay: $t_{\text{reject}} \leftarrow t_{\text{base}} + \mathcal{U}(0, t_{\max})$ Always return same error format regardless of failure cause
3) Ciphertext Sanitization:
   - Apply randomized padding: $\mathbf{c} \leftarrow \mathbf{c} + \mathbf{r}$ where $r_i \sim D_{\sigma_{\text{noise}}}$
   - Use rejection sampling: Accept plaintext with probability $\exp(-\|\mathbf{m}\|^2/2\sigma^2)$

4) Compiler-Level Protections:

1) Microarchitecture Hardening:
   - Insert serialization barriers: `lfence` before/after sensitive operations
   - Disable DVFS during crypto: Lock CPU frequency governor to performance mode
2) Code Generation Constraints:
   - Enforce `-fwrapv -fno-strict-aliasing` compiler flags
   - Use `__attribute__((optimize("-O1")))` for critical functions
3) Memory Protection:
   - Apply XOR-based memory wiping: $s_i \leftarrow s_i \oplus r_i$ before deallocation
   - Use guard pages: `mprotect(key_region, PROT_NONE)` after operations

5) Unified Masking Strategies:

1) Arithmetic Secret Sharing:
   - Split secrets: $s = s^{(1)} + s^{(2)} + s^{(3)} \mod q$ with refresh $s^{(i)} \leftarrow s^{(i)} + m^{(i)} - m^{(i-1)}$
   - Masked NTT: Compute $\widetilde{a} = \mathsf{NTT}(a + r)$ then correct $\mathsf{NTT}(a) = \widetilde{a} - \mathsf{NTT}(r)$
2) Blinding Transformations:
   - Input blinding: $A' = A \cdot D$ where $D = \mathrm{diag}(d_1, \ldots, d_n), d_i \in \mathbb{Z}_q^\times$
   - Output blinding: $\mathbf{z}' = \mathbf{z} + \mathbf{r}$ with $\|\mathbf{r}\|_\infty < \beta - \|\mathbf{z}\|_\infty$
3) Threshold Implementations:
   - Split non-linear functions: $f(x) = f_1(x^{(1)}) \oplus f_2(x^{(2)}) \oplus f_3(x^{(3)})$
   - Maintain non-completeness: f_i independent of $x^{(j)}$ for $i \neq j$

G. Summary

Overall there still are potential attacks in PQC they are however focussed on different aspects and seem to be less frequent, it seems that Falcon has the most potential for HFBs as it requires floating point arithmetic.

The full collection is accessible here[1], and a table summarizing the work done in this section is found here I.

VII. Testing Implementation

This section presents the extension of Google's Wycheproof framework to support post-quantum cryptographic (PQC) schemes. A foundational C test harness was developed during the first semester, providing minimal working support for signature verification and decapsulation. The current phase formalizes the generation of test vectors and schema integration, guided by the vulnerability taxonomy introduced earlier in the report. The aim is to produce Known Answer Tests (KATs), malformed vectors, and adversarial inputs for PQC schemes like Falcon and Kyber that can reveal subtle implementation flaws.

A. Overview of Implementation Structure

The implementation[2] is organized into the following directories:

- `implementations/` – Header files and core algorithm implementations
  - Headers: `falcon.h`, `mldsa.h`, `mlkem.h`
  - Implementation subdirectories: `falcon/`, `mldsa/`, `mlkem/`
- `parsing-gen/` – Test vector generation and parsing scripts
  - Falcon/Kyber scripts: `script_falc.py`, `script_kyb.py`
  - KAT directories: `KAT_falc/`, `KAT_kyb/`
- `schemas/` – JSON schema definitions for test vector validation
  - Examples: `falcon_sign_schema.json`, `mldsa_sign_schema.json`
- `tested-implementations/` – Validated algorithm implementations
- `unit/` – Unit tests for core functionality
  - Test files: `mlkem_test.c`, `mldsa_test.c`, `falcon_test.c`

[1] https://github.com/mattc-try/wycheproof-pqc/collection
[2] https://github.com/mattc-try/wycheproof-pqc/

TABLE I: Summary of Post-Quantum Cryptographic Vulnerabilities and Countermeasures

Vulnerability Type	Affected Schemes	Attack Mechanism	Countermeasures
Side-Channel (NTT)	Kyber, Dilithium	• Timing leaks from butterfly ops • DVFS frequency leaks (Kyber) • CPA on intermediates	• Constant-time Montgomery reduction • Randomized NTT sequences • Blinded coefficient access
	Falcon	• Mantissa Hamming weight (EM) • `fmod` timing leaks	• Constant-time FP reductions • Masked FFT operations
Fault Injection	Falcon, Dilithium	• PRNG state corruption • Keccak stuck-at faults • FFT coefficient errors	• Dual-rail execution • Keccak CRC32 checks • Reed-Solomon ECC
Power Analysis	Falcon, Dilithium, Kyber	• Rejection sampling loops • Sign-bit Hamming distance • Template attacks (Dilithium)	• Constant-time CDT sampling • Arithmetic secret sharing • Timing noise injection
Oracle Attacks	Kyber, Dilithium	• Early-abort timing (verification) • Plaintext-checking oracles	• Fixed-latency norm checks • Randomized response delays • Ciphertext sanitization
Compiler-Level Leaks	Kyber	• Variable-time `idiv` (gcc) • DVFS-induced side channels	• `-fwrapv` compiler flags • CPU frequency locking • Serialization barriers (`lfence`)

- `vect/` – Known Answer Test (KAT) validators
  - Test files: `v_mlkem.c`, `v_mldsa.c`, `v_falcon.c`
- `vectors/` – Test vector storage (JSON format)

B. Schema Generation and Adaptation

A dedicated schema file named `falcon_sign_schema.json` was developed to accommodate Falcon-based signature testing. The schema follows the Wycheproof model and includes the following structure:

- `algorithm`: a string indicating the algorithm variant, such as `Falcon-512`.
- `generatorVersion`: a version identifier to track the generator or script that produced the vector.
- `header`: optional documentation lines included as an array of strings.
- `notes`: a dictionary mapping test flags to human-readable explanations.
- `numberOfTests`: the total number of individual test cases.
- `schema`: fixed to `falcon_sign_schema.json` for validation purposes.
- `testGroups`: an array of `FalconSignTestGroup` objects.

Each `FalconSignTestGroup` defines:

- `type`: fixed to `FalconSign`.
- `privateKey`: a hex-encoded Falcon signing key used to produce the signatures.
- `tests`: a list of test cases of type

`FalconSignTestVector`.

Each `FalconSignTestVector` includes:

- `tcId`: an integer test case ID.
- `msg`: the input message in hexadecimal format.
- `sig`: the encoded signature in hexadecimal.
- `result`: either `valid` or `invalid`, specifying the expected output of the verifier.
- `comment`: optional string describing the purpose or origin of the test.
- `flags`: optional array of strings categorizing the test case (e.g., `bitflip`, `trailing-junk`, `malleability`).

C. Test Vector Generation Strategy

The process of constructing test vectors involved both automatic and manual techniques. The main objective was to include not only conformance vectors from NIST but also vectors designed to expose edge-case vulnerabilities.

The steps are as follows:

1) Baseline validation: Known Answer Tests were extracted from the official NIST PQC round 3 submissions. These included Falcon-512/1024 and ML-KEM vectors, parsed and converted into JSON.
2) Implementation bug reproduction: Using the Falcon reference implementation in PQClean, a known anomaly was reproduced where a mutated Falcon signature was still accepted as valid. The bug originated from slightly out-of-bound values in the compressed signature representation, possibly due to incorrect rejection sampling in Gaussian distribution or incomplete parser checks.
3) Input crafting and mutation: The test vector suite was expanded by modifying valid $(\text{message}, \text{signature}, \text{public key})$ tuples through:
   - bit-flipping bytes near the end of the signature
   - changing encodings to add redundant zero-padding
   - duplicating valid signatures and introducing single-byte offsets

   If the altered signature was still accepted, the result field was intentionally set to `invalid` to detect over-acceptance behavior.
4) Classification and metadata: Each vector was annotated with flags such as `over-accept`, `trailing-junk`, or `malleability` and given a descriptive comment to facilitate debugging.

A representative example is derived from a known issue in Falcon-512, where a signature generated by PQClean's reference implementation was modified by flipping bits in its compressed representation. The modified signature remained valid under verification, despite being non-identical to the canonical output.

This behavior results from floating-point instability during discrete Gaussian sampling and compression. Falcon's reliance on fused multiply-add (FMA) and FFT-based field operations makes it sensitive to deviations in the least significant bits. Minor differences in floating-point evaluations, even from instruction substitution or compiler reordering, can yield different but valid-looking signatures.

In the test, the same message and public key were used with a mutated signature:

```
// Original (valid)
verify(msg, sig, pk) == success

// Mutated (should be invalid)
verify(msg, sig_mutated, pk) == success
// incorrect
```

The mutated signature was encoded in the test vector JSON with identical `msg` and `pk`, altered `sig`, and `result` set to `"invalid"`. Flags included `bitflip`, `malleability`, and `non-determinism`. The test exposed Falcon's lack of strong unforgeability and highlighted the necessity of exact floating-point adherence.

D. Integration of Known Answer Tests

Both valid and invalid test vectors were compiled into structured JSON files. The valid cases verify correctness across Falcon-512, Falcon-1024, and Kyber variants. The invalid cases explore malformed inputs, short signatures, non-canonical encodings, and malformed public keys.

These vectors can be directly executed through the Wycheproof modular test runner with the following guarantees:

- Correct handling of boundary and malformed inputs
- Transparent origin tracking via comments and flags
- Compatibility with CI pipelines and multiple language backends

E. Planned Contributions

All test vector files, schema definitions, and test drivers are being prepared for adaptability to most libraries possible and following closely wycheproof work. Contributions include:

- A Falcon-focused signature test suite using malformed, edge-case, and legitimate inputs
- Schema extensions for ML-KEM and ML-DSA operations
- A framework for integrating fuzzing-derived vectors and differential testing across PQClean and liboqs backends

Additional work will target support for multivariate, code-based, and isogeny-based schemes, enabling broader PQC evaluation using a uniform vector testing methodology.

VIII. Conclusion

The transition to post-quantum cryptography (PQC) is a critical imperative necessitated by the existential threat quantum computing poses to classical cryptographic systems like RSA and ECC. While PQC schemes, including lattice-based (Kyber, Dilithium, Falcon), hash-based (SPHINCS+), code-based, multivariate, and isogeny-based algorithms, offer conjectured quantum resistance, their inherent novelty and reliance on complex mathematical structures (e.g., polynomial rings, lattice problems, hash forests) introduce distinct and challenging implementation vulnerabilities, termed "hard-to-find bugs" (HFBs). These HFBs are elusive, manifesting under rare conditions due to low trigger probability, algorithmic

subtlety, and cross-layer complexity, posing a significant risk of cryptographic failures, unauthorized data access, or private key leakage.

This study systematically characterizes PQC-specific HFBs, diverging significantly from classical cryptography's typical carry propagation or modular reduction errors, which are largely absent in PQC due to bounded coefficients and small moduli. New categories of vulnerabilities arise from polynomial coefficient management, Number-Theoretic Transform (NTT)-domain arithmetic, and compliance with error distribution specifications. Over 10 such vulnerabilities were identified, particularly within lattice-based cryptographic implementations like Kyber, Dilithium, and Falcon, highlighting their practical relevance.

Detailed analyses revealed common software-based attack vectors:

- Side-channel attacks on NTT operations, including timing leaks from butterfly operations, DVFS frequency leaks, and Correlation Power Analysis (CPA) on intermediate values, capable of recovering secret coefficients in Kyber and Dilithium. Falcon implementations are further susceptible to floating-point mantissa leakage and timing/power side channels related to trigonometric functions and modular reductions within its Gaussian sampler.
- Fault injection attacks can exploit deterministic signing processes in Falcon and Dilithium by corrupting PRNG states (e.g., ChaCha20 or Keccak-f), enabling secret key recovery via lattice reduction or sparse linear algebra.
- Power analysis on discrete Gaussian samplers, common in Falcon, Dilithium, and Kyber, leaks information through rejection sampling loop iterations and sign bit operations, allowing statistical recovery of sampled values or secret key bits.
- Plaintext-checking oracle attacks against Dilithium and Kyber exploit data-dependent verification timing, enabling binary search for secret coefficients through crafted inputs and observation of early-abort behavior.
- Compiler-level optimizations, particularly variable-time integer division (`idiv`) in Kyber, can introduce timing side channels, leading to secret key recovery through measurement of cycle counts correlated with sensitive values. Falcon's reliance on floating-point arithmetic also makes it vulnerable to rounding discrepancies across different hardware or compiler options, potentially revealing secret key constraints.

In response to these identified vulnerabilities, this report offers a suite of comprehensive and unified mitigation strategies applicable across various PQC schemes:

- Side-channel protection is achieved through constant-time arithmetic (e.g., Montgomery/Barrett reduction, unconditional subtraction-addition), blinded memory access, randomized NTT sequences, and leakage-resistant sampling techniques (e.g., constant-time CDT, timing noise injection).
- Fault injection mitigation involves computational redundancy (dual-rail execution, algebraic checksums), state verification (Keccak integrity checks, PRNG state duplication), and error-correcting codes (e.g., Reed-Solomon, Hamming codes) for critical data.
- Oracle attack prevention relies on ensuring constant-time verification processes (e.g., full norm checks, fixed-latency comparisons), response uniformization (randomized delays, consistent error formats), and ciphertext sanitization.
- Compiler-level protections are addressed by microarchitecture hardening (serialization barriers, CPU frequency locking) and code generation constraints (e.g., `-fwrapv`, memory protection techniques like XOR-based wiping and guard pages).
- Unified masking strategies apply arithmetic secret sharing, input/output blinding transformations, and threshold implementations for non-linear functions to protect secret values.

To systematically detect these elusive flaws, a tailored Known Answer Test (KAT) suite was developed by extending Google's Wycheproof framework for PQC primitives, specifically Falcon and Kyber. This involved designing new JSON schema definitions and a strategy for generating test vectors from NIST conformance cases, reproducing known implementation bugs (e.g., Falcon's acceptance of mutated signatures due to floating-point instabilities), and crafting adversarial inputs through bit-flipping, zero-padding, and single-byte offsets to expose edge-case vulnerabilities and detect over-acceptance behavior.

These contributions collectively establish a foundational framework for understanding, detecting, and mitigating implementation-level vulnerabilities in post-quantum cryptographic systems. By systematically characterizing PQC-specific HFBs, analyzing their exploitability across diverse attack mechanisms, and proposing robust countermeasures and advanced testing methodologies, this work provides a critical roadmap to ensure the secure and reliable deployment of quantum-safe cryptographic infrastructures in an evolving threat landscape.

REFERENCES

[1] Agence nationale de la sécurité des systèmes d'information. Recommendations for the use of post-quantum cryptography. Technical report, 2022.

[2] Daniel Apon et al. Efficient algorithms for supersingular isogeny diffie-hellman. In *Advances in Cryptology - CRYPTO 2019*, pages 572–601, 2019.

[3] Anja Becker et al. Information-set decoding for linear codes over fq. *Journal of Cryptology*, 30:809–830, 2017.

[4] Elwyn R. Berlekamp, Robert J. McEliece, and Henk C. A. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24(3):384–386, 1978.

[5] Daniel J. Bernstein. Cost analysis of hash collisions: Will quantum computers make sharcs obsolete? *SHARCS'09*, 2009.

[6] Daniel J. Bernstein et al. Sphincs: practical stateless hash-based signatures. In *Advances in Cryptology - EUROCRYPT 2015*, pages 368–397, 2015.

[7] Daniel J. Bernstein, Andreas Hülsing, Stefan-Lukas Gazdag, Johannes Kamp, Andreas Kühn, Tanja Lange, Ruben Niederhagen, Gregor Seiler, Jakub Szefer, and Peter Schwabe. Sphincs+: Submission to the nist post-quantum project. In *Submission to the NIST Post-Quantum Cryptography Standardization Project*, 2019. https://sphincs.org/data/sphincs+-r3-specification.pdf.

[8] Ward Beullens. Breaking rainbow takes a weekend on a laptop. *IACR Cryptology ePrint Archive*, 2021:214, 2021.

[9] Joppe W. Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from lwe. Cryptology ePrint Archive, Report 2016/659, 2016.
[10] Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals - kyber: A cca-secure module-lattice-based kem. In *2018 IEEE European Symposium on Security and Privacy (EuroSP)*, pages 353–367, 2018.
[11] Zvika Brakerski et al. Classical hardness of learning with errors. *Journal of the ACM*, 60(6):43, 2013.
[12] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. *Journal of the ACM*, 60(6):1–43, 2013.
[13] Johannes Buchmann et al. *Post-Quantum Cryptography*. Springer, 2011.
[14] Johannes A Buchmann, Denis Butin, Florian Göpfert, and Albrecht Petzoldt. Post-quantum cryptography: state of the art. *The New Codebreakers: Essays Dedicated to David Kahn on the Occasion of His 85th Birthday*, pages 88–108, 2016.
[15] Wouter Castryck and Thomas Decru. An efficient key recovery attack on sidh. *IACR Cryptology ePrint Archive*, 2022:975, 2022.
[16] L. Chen and T. Günther. Leakage analysis of negation in avr and arm implementations of falcon. In *Cryptographic Hardware and Embedded Systems (CHES) 2020*, volume 12100 of *LNCS*, pages 350–370, 2020.
[17] Luca De Feo. Mathematics of isogeny based cryptography. *arXiv preprint arXiv:1711.04062*, 2018.
[18] Jintai Ding et al. The rainbow signature scheme. In *Post-Quantum Cryptography*, pages 328–353, 2020.
[19] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-dilithium: A lattice-based digital signature scheme. In *Advances in Cryptology - ASIACRYPT 2018, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 238–265. Springer, 2018.
[20] A. Dupont et al. Power analysis of gaussian sampling in lattice signatures. *Journal of Cryptographic Engineering*, 11(3):211–228, 2022.
[21] European Telecommunications Standards Institute. Etsi ts 104 015: Quantum-safe hybrid key exchanges, 2023.
[22] Scott Fluhrer and David A. McGrew. Leighton-micali hash-based signatures. RFC 8554, Internet Engineering Task Force, April 2019.
[23] Pierre-Alain Fouque, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, Peter Schwabe, Damien Stehlé, and Alexandre Wallet. Falcon: Fast-fourier lattice-based compact signatures over ntru. In *Submission to the NIST Post-Quantum Cryptography Standardization Project*, 2018. https://falcon-sign.info/falcon.pdf.
[24] Philippe Gaborit, Adrien Hauteville, and Jean-Pierre Tillich. Hqc: A hamming quasi-cyclic public key encryption scheme. In *Post-Quantum Cryptography (PQCrypto 2017)*, volume 10346 of *Lecture Notes in Computer Science*, pages 43–58. Springer, 2017.
[25] Steven D. Galbraith. Lessons learned from the cryptanalysis of sidh. *Journal of Cryptology*, 35(4):45, 2022.
[26] F. Garcia and R. Patel. Template attacks on intermediate vectors in lattice signatures. In *Eurocrypt 2024, Part II*, volume 14045 of *LNCS*, pages 231–249, 2024.
[27] S. Ghosh and H. Kim. Floating-point discrepancies in ntru lattice signatures. Report 2023/456, Cryptology ePrint Archive, 2023. https://eprint.iacr.org/2023/456.
[28] Russ Housley et al. Implementing post-quantum cryptography in internet protocols. Technical report, Internet Engineering Task Force, 2023.
[29] International Organization for Standardization. Iso/iec jtc 1/sc 27 - it security techniques, 2023.
[30] ISO/IEC JTC 1/SC 27. Iso/iec 18033-4: Information technology - security techniques - encryption algorithms - part 4: Code-based mechanisms, 2023. Includes Classic McEliece.
[31] ISO/IEC JTC 1/SC 27. Iso/iec 18033-2: Information technology - security techniques - encryption algorithms - part 2: Asymmetric ciphers - amendment 1: Learning with errors, 2024. Includes FrodoKEM.
[32] Panos Kampanakis et al. Performance analysis of post-quantum cryptography in tls 1.3. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 1234–1248, 2023.
[33] Panos Kampanakis and Devin Sikeridis. Post-quantum authentication in tls 1.3: A performance study. In *NDSS*, 2020.
[34] E. O. Kiktenko et al. Quantum-safe cryptography: Foundations and challenges. *Symmetry*, 15(2):261, 2023.
[35] D. Kim and K. O'Connor. Differential fault analysis on deterministic lattice signatures. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(2):45–67, 2024.
[36] H. Kim and J. Park. Plaintext-checking oracle via dilithium side channels. *IEEE Security & Privacy*, 22(2):54–67, 2024.
[37] Seunghwan Kim, Jooyoung Lee, and Chaehoon Park. Multivariate quadratic solver in gf(31) with grover's algorithm. In *2022 IEEE International Conference on Quantum Computing*, pages 87–94, 2022.
[38] C. Lee and S. Omar. Dvfs side-channels in avx2 ntt: Recovering kyber secrets via frequency leakage. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(3):333–352, 2024.
[39] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *Journal of the ACM*, 60(6):1–35, 2013.
[40] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error-correcting codes*. Elsevier, 1977.
[41] Kalikinkar Mandal, Debasis Giri, and Ravi Shekhawat. Storage friendly provably secure multivariate identity-based signature from isomorphism of polynomials problem. In *SECRYPT 2021*, pages 595–602, 2021.
[42] Robert J. McEliece. A public-key cryptosystem based on algebraic coding theory. *DSN Progress Report*, 42-44:114–116, 1978.
[43] David A. McGrew, Michael Curcio, and Scott Fluhrer. Xmss: extended merkle signature scheme. RFC 8391, Internet Engineering Task Force, May 2018.
[44] National Institute of Standards and Technology. Falcon digital signature standard. Technical Report FIPS 206, U.S. Department of Commerce, 2024.
[45] National Institute of Standards and Technology. Module-lattice-based digital signature standard. Technical Report FIPS 204, U.S. Department of Commerce, 2024.
[46] National Institute of Standards and Technology. Module-lattice-based key-encapsulation mechanism standard. Technical Report FIPS 203, U.S. Department of Commerce, 2024.
[47] National Institute of Standards and Technology. Stateless hash-based digital signature standard. Technical Report FIPS 205, U.S. Department of Commerce, 2024.
[48] I. Nguyen and S. Gupta. Differential fault analysis on deterministic lattice signatures. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(1):95–117, 2024.
[49] National Institute of Standards and Technology. Post-quantum cryptography standardization, 2016–2024.
[50] Christian Paquin et al. Study of tls impacts on internet of things performance. In *IEEE Security and Privacy Workshops*, pages 140–145, 2019.
[51] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 333–342, 2009.
[52] Mohammad Sadeghi, Nasour Bagheri, and Mohammad Hadian. A new variant of the winternitz one time signature based on graded encoding schemes. *ISeCure*, 13(2):89–102, 2021.
[53] Peter W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134. IEEE, 1994.
[54] Abdulhadi Shoufan et al. A novel processor architecture for mceliece cryptosystem and fpga platforms. *IEEE Transactions on Computers*, 58(8):1043–1055, 2009.
[55] A. Smith and B. Jones. Fault attacks on keccak-based challenge generation in dilithium. In *CHES 2023*, volume 13374 of *LNCS*, pages 411–429, 2023.
[56] Daniel Smith-Tone et al. Vdoo: A multivariate encryption scheme, 2023.
[57] Douglas Stebila et al. Transitioning to a quantum-resistant public key infrastructure. In *Post-Quantum Cryptography*, pages 384–405, 2020.

APPENDIX

A. KEX PQC Hybrids

Hybrid schemes combine classical and post-quantum cryptographic algorithms to offer security against both classical and quantum adversaries. X-Wing is a hybrid key encapsulation mechanism (KEM) that combines the classical X25519 elliptic curve Diffie–Hellman key exchange with ML-KEM-768, a lattice-based KEM similar to Kyber.

1) Mathematical Formulation of X-Wing: Let λ be the security parameter. Define two KEMs:

$$(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{KeyGen}_1(1^\lambda), \quad (\mathsf{sk}_2, \mathsf{pk}_2) \leftarrow \mathsf{KeyGen}_2(1^\lambda),$$

where $\mathsf{KeyGen}_1, \mathsf{Encaps}_1, \mathsf{Decaps}_1$ implement X25519 and $\mathsf{KeyGen}_2, \mathsf{Encaps}_2, \mathsf{Decaps}_2$ implement ML-KEM-768. Encapsulation proceeds:

$$(\mathsf{ct}_i, K_i) \leftarrow \mathsf{Encaps}_i(\mathsf{pk}_i), \quad i = 1, 2, \quad \mathsf{ctx} = (\mathsf{ct}_1 \,\|\, \mathsf{ct}_2), \quad K_{\text{hyb}} = \text{KDF}(K_1 \| K_2),$$

where KDF is instantiated by SHA3-256 in extract-then-expand mode, producing a λ-bit output.

a) Correctness Condition: For honest parties A and B:

$$\mathsf{Decaps}_i(\mathsf{sk}_i^A, \mathsf{ct}_i^B) = K_i, \quad i = 1, 2,$$

thus

$$K_{\text{hyb}}^A = \text{KDF}(K_1 \,\|\, K_2) = K_{\text{hyb}}^B.$$

Decapsulation failure probabilities satisfy

$$\Pr[\mathsf{Decaps}_i(\cdot) \neq K_i] \leq \varepsilon_i, \quad \varepsilon_i \approx 2^{-128},$$

and hence

$$\Pr[K_{\text{hyb}}^A \neq K_{\text{hyb}}^B] \leq \varepsilon_1 + \varepsilon_2.$$

b) Security Guarantees: Assume each component is IND-CCA2 secure under its respective hardness assumption:

- X25519 is secure under the strong Diffie–Hellman assumption.
- ML-KEM-768 is secure under the Module-LWE assumption.
- SHA3-256 is modeled as a (quantum-)secure random oracle.

Then for any IND-CCA2 adversary $\mathcal{A}$,

$$\text{Adv}_{\text{XWing}}(\mathcal{A}) \leq \text{Adv}_{\mathsf{X25519}}(\mathcal{A}_1) + \text{Adv}_{\mathsf{ML\text{-}KEM}}(\mathcal{A}_2) + \delta_{\text{KDF}},$$

where δ_{KDF} is the negligible distinguishing advantage against SHA3-256.

a) Threat Considerations:

- Side-channel leakage mitigated by constant-time implementations and masking.
- Certificate authority compromise managed via short-lived certificates and transparency logs.
- Denial-of-service mitigated by rate-limiting and batch verification.

B. QKD PQC Hybrids

Quantum Key Distribution (QKD) provides information-theoretic key material but requires an authenticated classical channel for post-processing steps (basis sifting, error correction, privacy amplification). Post-quantum digital signatures serve as scalable, quantum-resistant authentication.

Let $\mathsf{SIG} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ be a PQC signature scheme, e.g., CRYSTALS-Dilithium or FALCON. For each classical message m:

$$\sigma = \mathsf{Sign}(\mathsf{sk}, m), \quad \mathsf{Verify}(\mathsf{pk}, m, \sigma) = \text{true}.$$

Two authentication strategies:

1) Multi-authentication Protocol: Sign each post-processing message individually:

$$\sigma_i = \mathsf{Sign}(\mathsf{sk}, m_i), \quad i = 1, \ldots, N.$$

2) Mono-authentication Protocol: Accumulate all transcripts $M = m_1 \,\|\, m_2 \,\|\, \cdots \,\|\, m_N$ and sign once:

$$\Sigma = \mathsf{Sign}(\mathsf{sk}, M).$$

Integrating PQC signatures into a PKI reduces key management from $\mathcal{O}(n^2)$ symmetric keys to $\mathcal{O}(n)$ certificates. Lattice-based schemes provide quantum-resistant authentication with signature sizes in the range 0.7–$4.6\,\text{kB}$.