

Complexity of Post-Quantum Cryptography in Embedded Systems and Its Optimization Strategies

Omar Alnaseri*, Yassine Himeur[†], Shadi Atalla[†] and Wathiq Mansoor[†]

*Department of Electrical Engineering, DHBW University, Ravensburg, Germany

[†]College of Engineering and Information Technology, University of Dubai, Dubai, United Arab Emirates

Abstract—With the rapid advancements in quantum computing, traditional cryptographic schemes like Rivest-Shamir-Adleman (RSA) and elliptic curve cryptography (ECC) are becoming vulnerable, necessitating the development of quantum-resistant algorithms. The National Institute of Standards and Technology (NIST) has initiated a standardization process for PQC algorithms, and several candidates, including CRYSTALS-Kyber and McEliece, have reached the final stages. This paper first provides a comprehensive analysis of the hardware complexity of post-quantum cryptography (PQC) in embedded systems, categorizing PQC algorithms into families based on their underlying mathematical problems: lattice-based, code-based, hash-based and multivariate / isogeny-based schemes. Each family presents distinct computational, memory, and energy profiles, making them suitable for different use cases. To address these challenges, this paper discusses optimization strategies such as pipelining, parallelization, and high-level synthesis (HLS), which can improve the performance and energy efficiency of PQC implementations. Finally, a detailed complexity analysis of CRYSTALS-Kyber and McEliece, comparing their key generation, encryption, and decryption processes in terms of computational complexity, has been conducted.

Index Terms—Post-Quantum Cryptography, Embedded Systems, Hardware Complexity

I. INTRODUCTION

Post-quantum cryptography (PQC) aims to develop cryptographic algorithms that are secure against attacks from quantum computers. With rapid advancements in quantum computing, traditional cryptographic schemes such as Rivest-Shamir-Adleman (RSA) and elliptic curve cryptography (ECC) are becoming vulnerable. PQC algorithms are broadly categorized into families based on their underlying mathematical problems, each posing distinct challenges for implementation in embedded systems. These families include lattice-based, code-based, hash-based, and multivariate/isogeny-based schemes, each with unique computational, memory, and energy profiles.

Lattice-based algorithms derive security from hard problems like the shortest vector problem (SVP) or learning with errors (LWE). Schemes such as Kyber (key encapsulation) and CRYSTALS-Dilithium (digital signatures) rely on polynomial arithmetic and the number theoretic transform (NTT) for efficient polynomial multiplication. While NTT speeds up computations, it introduces hardware overhead due to

modular arithmetic and parallel operations. Ducas et al. [1] showed NTT optimizations reduce latency by up to 40% on embedded platforms. However, lattice-based algorithms still require moderate memory (1-3 kB RAM for keys) and scalable parameters, making them demanding for limited devices. Kannwischer et al. [2] demonstrated Kyber on ARM Cortex M4, using less than 10 kB RAM, although it is less energy efficient than hash-based schemes.

Code-based algorithms, such as McEliece and BIKE, rely on decoding random linear codes. They are computationally lightweight, using matrix multiplications and sparse linear algebra, but suffer from large key sizes (often over 1 MB). Chou et al. [3] reduced McEliece keys by 50% on STM32 microcontrollers, but compressed keys still overload flash storage, limiting their use in memory-constrained devices.

Hash-based algorithms, such as SPHINCS+ and XMSS [4], use cryptographic hash functions (e.g. SHA-3 or SHAKE-256). They are computationally simple, relying on iterative hashing, and are lightweight and parallelizable. Bernstein et al. [5] showed SPHINCS+ runs on 8-bit AVR microcontrollers with just 2 kB RAM, making it suitable for energy-constrained IoT devices. However, stateless schemes like SPHINCS+ produce large signatures (up to 41 KB), while stateful schemes (e.g., XMSS) require nonvolatile memory (NVM) for state tracking. Bernstein et al. [6] noted that parallelizing hash chains improves throughput but increases area overhead, limiting cost-sensitive applications.

Multivariate and isogeny-based algorithms are niche categories with limited embedded adoption. Multivariate schemes like Rainbow involve solving nonlinear polynomial equations, leading to computationally intensive operations and large keys (often over 100 kB). Isogeny-based schemes, such as SIKE [7], use complex elliptic curve isogeny computations that are sequential and hard to accelerate. Koziel et al. [8] showed that SIDH FPGA implementations require significant area and power, even with optimizations. Table I compares cryptographic families in computational complexity, memory footprint, energy efficiency, and flexibility.

The National Institute of Standards and Technology (NIST) has narrowed the candidates to a few finalists, including CRYSTALS-Kyber and McEliece [9]. Nonetheless, embedded systems, characterized by their limited com-

TABLE I: Comparison of Cryptographic Families

Family	Computational Complexity	Memory Footprint	Energy Efficiency	Implementation Flexibility
Lattice-Based	Medium-High (NTT)	Medium (1–3 kB RAM)	Medium	High (scalable parameters)
Code-Based	Low	Very High (>1 MB)	Low	Low (fixed key sizes)
Hash-Based	Low	Low (<2 kB RAM)	Very High	Medium (state management)
Multivariate	Very High	High (100 kB)	Very Low	Low
Isogeny-Based	Very High	Medium (10 kB)	Very Low	Low

putational power and memory, present unique challenges for PQC implementation. Achieving successful deployment of these candidates on embedded systems necessitates a comprehensive analysis of the hardware complexity associated with these PQC algorithms. Thus, this paper conducts an exhaustive examination of the hardware complexity involved in deploying PQC within such systems, with a particular emphasis on CRYSTALS-Kyber and McEliece. This paper makes several key contributions to the field of PQC in embedded systems:

- **Comprehensive analysis:** The paper provides a detailed analysis of the hardware complexity of PQC algorithms, categorizing them into families based on their underlying mathematical problems, and discussing their computational, memory, and energy profiles.
- **Optimization strategies:** The paper explores optimization strategies such as pipelining, parallelization, and high-level synthesis (HLS) to improve the performance and energy efficiency of PQC implementations in embedded systems.
- **Complexity comparison:** It offers a detailed complexity analysis of two leading PQC candidates, CRYSTALS-Kyber and McEliece, comparing their key generation, encryption, and decryption processes in terms of computational complexity, memory footprint, and energy efficiency.

II. OPTIMIZATION STRATEGIES

In this section, we propose optimization strategies that improve the performance of PQC algorithms in embedded systems by focusing on techniques that improve computational efficiency and resource management. We will explore methods such as pipelining, parallelization, and efficient use of hardware resources to maximize algorithmic throughput and reduce execution time.

A. Pipelining and Parallelization

Pipelining involves breaking down a complex task into a series of smaller tasks, each performed at a different stage of a pipeline [10]. This allows for parallel processing, where multiple tasks can be executed simultaneously, reducing the overall processing time. Pipelining can be applied to various stages of the cryptographic process, such as key generation, signature generation, and signature verification [11]. For

example, the CRYSTALS-Dilithium scheme uses a pipelined processing method to reduce both storage requirements and processing time [10]. Similarly, the Picnic digital signature scheme uses a pipelined approach to optimize its hardware implementation, resulting in a significant reduction in clock cycle count and energy consumption [11].

Parallelization, on the other hand, involves dividing a task into smaller subtasks that can be executed concurrently by multiple processing units [12]. This can be achieved using parallel architectures, such as multi-core processors or specialized accelerators. For instance, the RISQ-V architecture [12] integrates tightly coupled accelerators directly into the processing pipeline to speed up lattice-based cryptography. The accelerators include an arithmetic unit for vectorized modular arithmetic and NTT operations, a vectorized modular multiply accumulate unit, a Keccak accelerator for the pseudo-random bit generation, and a binomial sampling unit for the generation of binomially distributed samples.

Pipelining and parallelization can be combined to further improve performance. For example, the design of the "coding for energy reduction with multiple encryption techniques" (CERMET) architecture incorporates both pipelining and process parallelization to improve efficiency [13]. The system operates fully pipelined, ensuring that no throughput is lost compared to a conventional cryptographic system, and maintains throughput despite additional data processing steps. However, pipelining and parallelization can also introduce additional complexity and overhead. For example, the polynomial factorization method used in parallel quantum signal processing can reduce the depth of the query by a factor $O(k)$, but it comes with an increased number of measurements $O(\text{poly}(d)2^{O(k)})$ [14], where k is the module rank.

B. High-Level Synthesis (HLS)

It automates design, creating hardware from algorithm descriptions. It optimizes post-quantum cryptography (PQC) for better performance, energy efficiency, and security. A hybrid HLS strategy combines state-based and performance-driven approaches, using periodic state machine models for precise timing and reduced energy use [15]. Another HLS optimization strategy is the use of a hierarchical post-route quality of results (QoR) prediction approach. This approach estimates latency and post-route resource usage from C/C++ programs and uses a graph construction method to represent

the control and data flow graph of source code and the effects of HLS pragmas. The approach also uses a hierarchical graph neural network (GNN) training and prediction method to capture the impact of loop hierarchies [16]. However, HLS optimizations can also affect the security and reliability of cryptographic implementations. For example, HLS optimizations can compromise the properties of countermeasures implemented using HLS, such as masking and hiding countermeasures. Therefore, secure circuit designers should be careful when using an HLS flow to integrate SCA countermeasures [17].

C. Algorithmic Optimizations

Optimizing the algorithms themselves can also reduce the hardware complexity. For example, using more efficient mathematical techniques can help reduce computational and memory requirements. Difference optimization strategies can be employed to improve the performance of PQC algorithms, such as:

1) *Hybrid approach*: combines quantum key distribution (QKD) with PQC for authentication purposes [18]. This can be particularly useful for protecting highly loaded communications links at a distance, where intermediate nodes may not be necessary. Additionally, standardization processes, such as those led by the NIST, can help identify and standardize post-quantum algorithms for stateless digital signatures and key encapsulation mechanisms/public key encryption.

2) *Signature lifting*: allows users who failed to migrate to PQC in time to still use pre-quantum signature schemes while protecting against quantum attacks [19]. This can be achieved by lifting a deployed pre-quantum signature scheme satisfying a certain property to a post-quantum signature scheme that uses the same keys.

3) *Quantum approximations*: can also be beneficial for optimizing PQC algorithms. For example, a quantum mean value approximation can be used to approximate the density of the lattice basis, which can be used to improve the performance of lattice-based cryptography [20].

4) *Quantum binary field multiplication*: can optimize PQC operations, which can achieve a Toffoli depth of one for any field size, making it more efficient for quantum cryptanalysis of ECC [21].

Algorithm 1 summarizes the proposed process to optimize the implementation of PQC in embedded systems, specifically for the CRYSTALS-Kyber and McEliece schemes. It consists of key generation, encryption, and decryption. Kyber constructs a public key using a polynomial matrix, while McEliece employs a Goppa code with scrambling and permutation matrices. The encryption and decryption processes involve modular arithmetic and error correction. The algorithm integrates pipelining, high-level synthesis, modular reduction, and memory optimization to enhance efficiency while maintaining security in embedded systems.

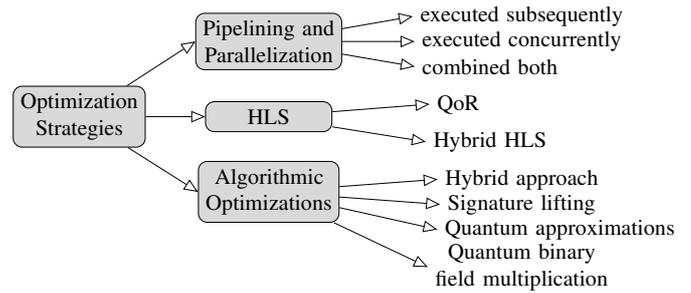


Fig. 1: Optimization Strategies

III. COMPLEXITY ANALYSIS

A. CRYSTALS-Kyber

CRYSTALS-Kyber is a lattice-based cryptosystem that relies on the module learning with errors (Module-LWE) problem. The security of Kyber is based on the difficulty of solving the Module-LWE problem, even for quantum computers, making it a strong candidate for post-quantum cryptography. The Kyber is a key encapsulation mechanism (KEM) that is part of the "cryptographic suite for algebraic lattices" (CRYSTALS) suite, which is designed to be secure against quantum computers. It operates in the ring $R_q = \mathbb{Z}_q[X]/(X^n + 1)$, where q is a prime modulus, n is the degree of polynomial, typically a power of 2, and $X^n + 1$ is the irreducible polynomial defining the ring [22]. To analyze the complexity, the operations primarily involve matrix-vector and vector-vector operations over polynomial rings. A breakdown of key operations of the kyber and their complexities is

- The key generation process involves first generate matrix $A \in R_q^{k \times k}$, where k is the security parameter, typically 2, 3, or 4. Then generate a secret key vector $s \in R_q^k$ from a centered binomial distribution η . And finally generate public key $t = A \cdot s + e$, where $e \in R_1^k$ is a small error vector sampled from the same distribution η , which is a discrete Gaussian-like distribution. This operation has a complexity of $O(k^2 \cdot n)$, where k is the module rank, i.e. 2, 3, or 4, and n is the polynomial degree, e.g. 256. Therefore the FLOPs for this matrix-vector multiplication is $2k^2n$.
- The encryption process involves first generate randomness $r \in R_q^k$ from the distribution η , then compute ciphertext (u, v) , as $u = A^T \cdot r + e_1$, and $v = t^T \cdot r + e_2 + \text{encode}(m)$, where $e_1 \in R_q^k$ and $e_2 \in R_q$ are small error, and $\text{encode}(m)$ is the encoded message m . Each matrix-vector multiplication in encryption has a complexity of $O(k^2 \cdot n)$, which results in a total of $2k^2n$ FLOPs.
- The decryption process involves first recovering the message m by computing $v - s^T \cdot u$, which should be close to $\text{encode}(m)$ due to small errors. This is

Algorithm 1: Optimized Post-Quantum Cryptography Implementation in Embedded Systems

Input: Security parameter k , Polynomial degree n , PQC scheme (Kyber or McEliece)

Output: Optimized Key Generation, Encryption, and Decryption

```
/* Step 1: Key Generation */
if Scheme == Kyber then
  Generate random matrix  $A \in R_q^{k \times k}$ ;
  Sample secret key vector  $s \in R_q^k$  and error vector
   $e \in R_q^k$ ;
  Compute public key:  $t = A \cdot s + e$ ;
end
else if Scheme == McEliece then
  Choose Goppa code parameters  $(n, k, t)$ ;
  Generate scrambling matrix  $S \in \mathbb{F}_2^{k \times k}$  and
  permutation matrix  $P \in \mathbb{F}_2^{n \times n}$ ;
  Compute public key:  $G' = S \cdot G \cdot P$ ;
end
/* Step 2: Encryption */
if Scheme == Kyber then
  Generate randomness  $r \in R_q^k$  and error terms
   $e_1, e_2$ ;
  Compute ciphertext:  $u = A^T \cdot r + e_1$ ,
   $v = t^T \cdot r + e_2 + \text{encode}(m)$ ;
end
else if Scheme == McEliece then
  Compute codeword:  $c = m \cdot G'$ ;
  Generate random error vector  $e \in \mathbb{F}_2^n$  with
  Hamming weight  $\leq t$ ;
  Compute ciphertext:  $y = c + e$ ;
end
/* Step 3: Decryption */
if Scheme == Kyber then
  Recover message:  $m = \text{decode}(v - s^T \cdot u)$ ;
end
else if Scheme == McEliece then
  Apply permutation:  $y' = y \cdot P^{-1}$ ;
  Decode using Goppa decoding algorithm to
  recover  $c'$ ;
  Recover message:  $m = c' \cdot G^{-1} \cdot S^{-1}$ ;
end
/* Optimization Strategies */
foreach Optimization technique in [Pipelining, HLS,
  Modular Reduction, Memory Optimization] do
  Apply technique to relevant PQC operation;
end
return Optimized Key Generation, Encryption, and
  Decryption;
```

an operation of the inner product with a complexity of $O(k \cdot n)$. This results in $2kn$ FLOPS.

Overall, Kyber is efficient in terms of key generation and decryption, with smaller key sizes and lower computational overhead, making it suitable for constrained environments.

B. McEliece

The McEliece cryptosystem is one of the earliest public-key cryptosystems, proposed by Robert McEliece in 1978 [23]. It is based on error-correcting codes, specifically binary Goppa codes, and is considered a strong candidate for post-quantum cryptography because of its resistance to attacks by quantum computers. It relies on the hardness of decoding a random linear code, which is a well-known problem in coding theory. The key components are: (1) linear codes C of length n and dimension k over a finite field \mathbb{F}_q are a subspace k of dimensions of \mathbb{F}_q^n . It can be represented by a generator matrix G of size $k \times n$. And (2) Goppa codes, which is a specific class of linear codes with efficient decoding algorithms. Goppa codes are used in McEliece because they allow for efficient error correction. The security of McEliece relies on the fact that decoding a random linear code is a hard problem, even for quantum computers. To analyze the complexity of McEliece operations, it involves matrix multiplications, inversions, and error correction. A breakdown of its key operations and their complexities is provided below.

- The key generation process involves first choosing a Goppa code C with parameters (n, k, t) , which are the length, dimension of the codes, and error correction ability, respectively. Then generate a random scrambling matrix S of $k \times k$ over \mathbb{F}_2 , and generate a random permutation matrix P of $n \times n$. Finally, compute the transformed generator matrix $G' = S \cdot G \cdot P$, so the public key is (G', t) , where t is the error correction capability. This matrix-matrix multiplication has a complexity of $O(n^3)$, where n is the length of the code. Thus, the number of FLOPs is $2n^3$.
- The encryption process involves encrypting a message $m \in \mathbb{F}_2^k$ by computing the codeword $c = m \cdot G'$ and generating a random error vector $e \in \mathbb{F}_2^n$ with Hamming weight $wt(e) \leq t$, and finally the ciphertext is calculated $y = c + e$. This matrix vector multiplication has a complexity of $O(n^2)$, therefore the number of FLOPs is $2n^2$.
- The decryption process starts with decoding the ciphertext y applying the permutation $y' = y \cdot P^{-1}$, and using the efficient decoding algorithm for the Goppa code to correct errors in y' and recovering the codeword c' by computing $m' = c' \cdot G^{-1}$, where G^{-1} is the inverse of the generator matrix G . Then apply the inverse of the scrambling matrix $m = m' \cdot S^{-1}$, where m is the decrypted message. This also has a complexity of $O(n^2)$ for efficient decrypting algorithms.

C. Comparison of Complexities

CRYSTALS-Kyber and McEliece differ significantly in their mathematical foundations and computational complexities. Kyber, based on lattice problems, is more efficient in terms of key generation and decryption, with smaller key sizes and lower computational overhead. This makes it particularly suitable for constrained environments and applications where key size and computational efficiency are critical. On the other hand, McEliece, based on coding theory, is efficient for encryption but suffers from large key sizes and higher key generation complexity. Based on Table II, McEliece has a higher complexity compared to Kyber by key generation. This is because McEliece involves matrix-matrix multiplications, which are more expensive than the matrix-vector operations in Kyber. In the encryption process, McEliece is slightly more efficient compared to Kyber, as it only requires matrix-vector multiplication. Kyber is more efficient than McEliece for decryption, as it requires only inner product operations. In terms of parameter sizes, Kyber typically uses smaller parameters, like $n = 256, k = 2, 3, 4$, while McEliece uses larger parameters, such as $n = 1024$. This makes Kyber more efficient in practice for key sizes and computational overhead.

TABLE II: Complexity Comparison Between CRYSTALS-Kyber and McEliece

Operation	CRYSTALS-Kyber	McEliece
Key Generation	$O(k^2n)$	$O(n^3)$
Encryption	$O(k^2n)$	$O(n^2)$
Decryption	$O(kn)$	$O(n^2)$

IV. NUMERICAL ANALYSIS COMPARISON

The numerical analysis is conducted employing the parameters specified in Table III and Table IV. Kyber is characterized by three distinct security levels, each associated with a specific set of parameters. Similarly, McEliece is characterized by a variety of parameter sets that are determined based on the code length and the error-correcting capability.

TABLE III: Kyber Security Levels and Parameters

Security Level	Module Rank (k)	Polynomial Degree (n)	Key Size (Bytes)	Ciphertext Size (Bytes)
Kyber512	2	256	800	768
Kyber768	3	256	1184	1088
Kyber1024	4	256	1568	1568

TABLE IV: McEliece Security Levels and Parameters

Security Level	Code Length (n)	Error-Correcting Capability (t)	Key Size (Bytes)	Ciphertext Size (Bytes)
348864	3488	64	261,120	128
460896	4608	96	524,160	188
6688128	6688	128	1,044,480	240

Fig. 2 compares the key sizes of CRYSTALS-Kyber and McEliece across different security levels (128-bit, 192-bit,

256-bit). The key size is a critical metric as it influences both the storage requirements and transmission overhead within cryptographic systems. CRYSTALS-Kyber has significantly smaller key sizes compared to McEliece. For example, Kyber512 (128-bit security) has a key size of 800 bytes, while McEliece-348864 (128-bit security) has a key size of 261,120 bytes. As the security level increases, the key sizes for both grow. However, the key sizes of McEliece continue to be several orders of magnitude greater than those of Kyber. Therefore, Kyber is more suitable for applications with limited storage or bandwidth, such as IoT devices or mobile communication.

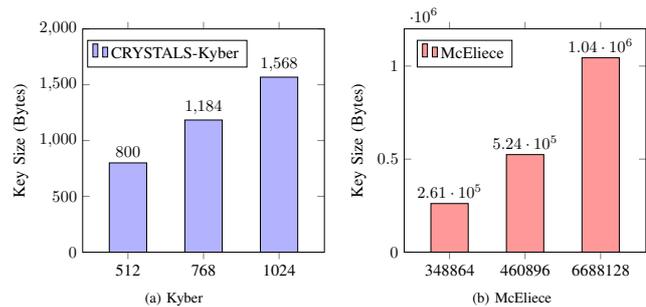


Fig. 2: Key Size

Fig. 3 compares the computational complexity of CRYSTALS-Kyber and McEliece in terms of FLOP counts for key generation, encryption, and decryption. FLOP counts provide a measure of the computational effort required for each operation. CRYSTALS-Kyber has much lower FLOP counts for all operations compared to McEliece. For example, Kyber-512 requires 2048 FLOPs for key generation, while McEliece-348864 requires 8.510^{10} FLOPs. Encryption and decryption in Kyber are also significantly faster, with FLOP counts in the thousands, compared to FLOP counts of McEliece in the millions or billions. Kyber is more efficient in terms of computational resources, making it better suited for resource-constrained environments or real-time applications.

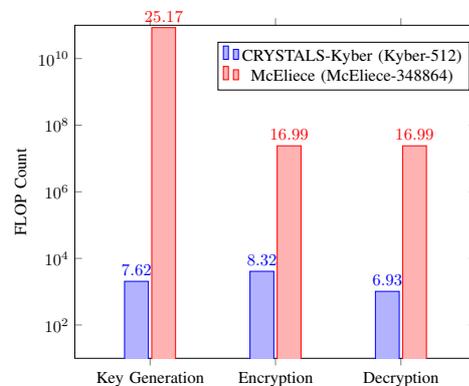


Fig. 3: FLOP Count

Fig. 4 compares the ciphertext sizes of CRYSTALS-Kyber and McEliece at different security levels. The size of the cryptotext is important because it affects the amount of data that must be transmitted during encryption. CRYSTALS-Kyber has larger ciphertext sizes compared to McEliece. For example, Kyber512 has a ciphertext size of 768 bytes, while McEliece-348864 has a ciphertext size of 128 bytes. However, the difference in ciphertext sizes is much smaller than the difference in key sizes. Although McEliece has smaller ciphertexts, its large key sizes and high computational complexity make it less practical for many applications. The marginally larger ciphertexts of Kyber are counterbalanced by its reduced key sizes and diminished computational overhead.

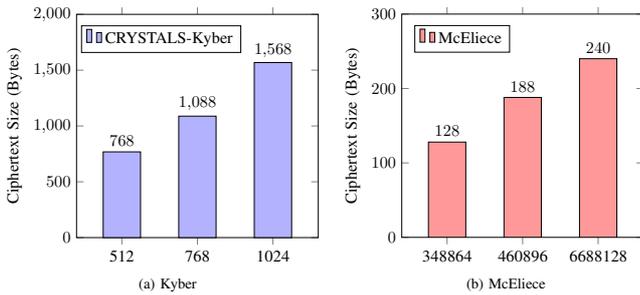


Fig. 4: Ciphertext Size

V. CONCLUSION

The paper examines the implementation of post-quantum cryptography (PQC) in embedded systems constrained by computational power, memory, and energy. It categorizes PQC algorithms into families: lattice-based, code-based, hash-based, and multivariate/isogeny-based schemes, each with unique challenges. Lattice-based schemes like CRYSTALS-Kyber require moderate memory and significant hardware for polynomial arithmetic. Code-based schemes like McEliece have large key sizes. Hash-based schemes such as SPHINCS+ have simple computation but large signatures, while multivariate/isogeny-based schemes demand too many resources. Optimization through pipelining, parallelization, and high-level synthesis can improve performance and energy efficiency while balancing security. CRYSTALS-Kyber suits constrained environments for key generation and decryption, unlike McEliece. Future research should explore new techniques and memory improvements to reduce hardware complexity, as quantum computing develops.

REFERENCES

- [1] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, “Crystals-dilithium algorithm specifications and supporting documentation,” 2017.
- [2] M. J. Kannwischer, M. Krausz, R. Petri, and S.-Y. Yang, “pqm4: Benchmarking nist additional post-quantum signature schemes on microcontrollers,” *Cryptology ePrint Archive*, 2024.
- [3] M.-S. Chen and T. Chou, “Classic mceliece on the arm cortex-m4,” *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 125–148, 2021.
- [4] J. A. Buchmann, E. Dahmen, and A. Hülsing, “Xmss - a practical forward secure signature scheme based on minimal security assumptions,” *IACR Cryptol. ePrint Arch.*, vol. 2011, p. 484, 2011.
- [5] D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe, “The sphincs+ signature framework,” in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, 2019, pp. 2129–2146.
- [6] D. J. Bernstein, S. Kölbl, S. Lucks, P. M. C. Massolino, F. Mendel, K. Nawaz, T. Schneider, P. Schwabe, F.-X. Standaert, Y. Todo *et al.*, “Gimli: a cross-platform permutation,” in *Cryptographic Hardware and Embedded Systems—CHES 2017: 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*. Springer, 2017, pp. 299–320.
- [7] R. Azarderakhsh, M. Campagna, C. Costello, L. D. Feo, B. Hess, A. Jalali, D. Jao, B. Koziel, B. LaMacchia, P. Longa *et al.*, “Supersingular isogeny key encapsulation,” *Submission to the NIST Post-Quantum Standardization project*, vol. 152, pp. 154–155, 2017.
- [8] B. Koziel, R. Azarderakhsh, M. M. Kermani, and D. Jao, “Post-quantum cryptography on fpga based on isogenies on elliptic curves,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 1, pp. 86–99, 2016.
- [9] M. Bandaru, S. E. Mathe, and C. Watanapanich, “Evaluation of hardware and software implementations for nist finalist and fourth-round post-quantum cryptography kems,” *Computers and Electrical Engineering*, vol. 120, p. 109826, 2024.
- [10] C. Zhao, N. Zhang, H. Wang, B. Yang, W. Zhu, Z. Li, M. Zhu, S. Yin, S. Wei, and L. Liu, “A compact and high-performance hardware architecture for crystals-dilithium,” *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2022, pp. 270–295, 2021.
- [11] G. Liu, K. Jia, P. Wei, and L. Ju, “High-performance hardware implementation of mpcith and picnic3,” *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2024, pp. 190–214, 2024.
- [12] T. Fritzmann, G. Sigl, and M. J. Sepúlveda, “Risq-v: Tightly coupled risc-v accelerators for post-quantum cryptography,” *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 446, 2020.
- [13] J. Woo, V. A. Vasudevan, B. Z. Kim, A. Cohen, R. G. L. D’Oliveira, T. Stahlbuhk, and M. M’edard, “Cernet: Coding for energy reduction with multiple encryption techniques - it’s easy being green,” *ArXiv*, vol. abs/2308.05063, 2023.
- [14] J. M. Martyn, Z. M. Rossi, K. Z. Cheng, Y. Liu, and I. Chuang, “Parallel quantum signal processing via polynomial factorization,” 2024.
- [15] Y. Liao, T. Adegbija, and R. L. Lysecky, “A high-level synthesis approach for precisely-timed, energy-efficient embedded systems,” *ArXiv*, vol. abs/2404.14769, 2022.
- [16] M. Gao, J. Zhao, Z. Lin, and M. Guo, “Hierarchical source-to-post-route qor prediction in high-level synthesis with gnns,” *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–6, 2024.
- [17] A.-A. Koufopoulou, K. Xevgeni, A. Papadimitriou, M. Psarakis, and D. Hély, “Security and reliability evaluation of countermeasures implemented using high-level synthesis,” *2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pp. 1–8, 2022.
- [18] S. E. Yunakovsky, M. Kot, N. O. Pozhar, D. Nabokov, M. A. Kudinov, A. Guglya, E. O. Kiktenko, E. Kolycheva, A. Borisov, and A. K. Fedorov, “Towards security recommendations for public-key infrastructures for production environments in the post-quantum era,” *EPJ Quantum Technology*, vol. 8, 2021.
- [19] O. Sattath and S. Wyborski, “Protecting quantum procrastinators with signature lifting: A case study in cryptocurrencies,” *ArXiv*, vol. abs/2303.06754, 2023.
- [20] D. Joseph, A. J. Martinez, C. Ling, and F. Mintert, “Quantum mean-value approximator for hard integer-value problems,” *Physical Review A*, 2021.
- [21] K. B. Jang, W. Kim, S. Lim, Y. L. Kang, Y. Yang, and H. Seo, “Quantum binary field multiplication with optimized toffoli depth and

extension to quantum inversion,” *Sensors (Basel, Switzerland)*, vol. 23, 2023.

- [22] F. R. Ghashghaei, Y. Ahmed, N. Elmrabit, and M. Yousefi, “Enhancing the security of classical communication with post-quantum authenticated-encryption schemes for the quantum key distribution,” *Comput.*, vol. 13, p. 163, 2024.
- [23] R. J. McEliece, “A public-key cryptosystem based on algebraic,” *Coding Thv*, vol. 4244, pp. 114–116, 1978.