IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2023.0322000

NIST Post-Quantum Cryptography Standards: A Comprehensive Review of Theoretical Foundations and Implementations

QUANG DANG TRUONG[1], HIEN NGUYEN[2], (Graduate Student Member, IEEE), TUY TAN NGUYEN[3], (Senior Member, IEEE), AND HANHO LEE[1], (Senior Member, IEEE)

[1]Department of Electrical and Computer Engineering, Inha University, Incheon 22212, Korea (e-mail: dangquang@inha.edu, hhlee@inha.ac.kr).
[2]School of Informatics, Computing, and Cyber Systems, Northern Arizona University, Flagstaff, AZ 86011, USA (e-mail: tn598@nau.edu).
[3]Department of Electrical and Computer Engineering, Center for Advanced Power Systems, FAMU-FSU College of Engineering, Florida State University, Tallahassee, FL 32310, USA (e-mail: tuy.nguyen@fsu.edu).

Corresponding author: Hanho Lee (e-mail: hhlee@inha.ac.kr).

This work was supported by the IITP (Institute of Information & Communications Technology Planning & Evaluation)-ITRC (Information Technology Research Center) grant funded by the Korea government (Ministry of Science and ICT, MSIT) (RS-2021-II212052), and in part by the Commercialization Promotion Agency for Research and Development Outcomes (COMPA) funded by MSIT (RS-2025-02219156).

ABSTRACT The transition to post-quantum cryptography (PQC) marks a pivotal shift in ensuring digital security, prompted by the potential of quantum computers to compromise classical systems such as Rivest-Shamir-Adlema and elliptic-curve cryptography. In response, NIST has standardized three foundational PQC algorithms: Module-lattice-based Key-encapsulation Mechanism for key establishment; Module-lattice-based Digital Signature, and Stateless Hash-based Digital Signature algorithms for digital signatures. Meanwhile, FALCON and Hamming Quasi-cyclic (HQC) schemes, both selected as finalists, are expected to join the standards soon. This paper presents a comprehensive survey of these NIST-selected PQC standards, with a dual focus on software implementations and hardware architecture designs. We analyze their mathematical frameworks, distinctive features, and optimization strategies related to performance, security, and resource efficiency. The software review examines algorithmic complexity, memory usage, and programming considerations, while the hardware review discusses FPGA and ASIC implementations, emphasizing modular arithmetic, polynomial operations, and resource efficiency challenges. A comparative analysis highlights the strengths and trade-offs of each algorithm, offering insights into their applicability across various platforms-from resource-constrained internet of things devices to high-performance computing environments. This study provides a foundational understanding of NIST's selected PQC standards and their practical deployment in securing the post-quantum era.

INDEX TERMS Post-quantum cryptography (PQC), key-encapsulation mechanism (KEM), digital signature algorithm (DSA), hardware architecture design, software optimization.

I. INTRODUCTION

OVER the past several years, quantum computing has made steady and significant progress toward practical, large-scale implementation. If realized, quantum computers would enable algorithms such as Shor's algorithm [1] to efficiently solve problems like integer factorization and discrete logarithms over finite fields and elliptic curves. This would break widely used cryptographic schemes such as Rivest-Shamir-Adleman (RSA) and elliptic curve cryptography (ECC), thereby undermining the security of many public-key cryptosystems that rely on the hardness of these problems. Although large-scale quantum computers are not yet available, recent advancements suggest that once they become practical, today's cryptographic standards could be rendered obsolete in a relatively short timeframe.

To address this threat, the U.S. National Institute of Standards and Technology (NIST) launched the post-quantum cryptography (PQC) standardization process in 2016 [2]. A total of 82 candidate algorithms, each based on different mathematical hardness assumptions, were submitted and evaluated based on three main criteria: (1) security, (2) cost and performance, and (3) algorithmic and implementation characteristics. After four rounds of evaluation and analysis, NIST selected five algorithms for standardization. Three of

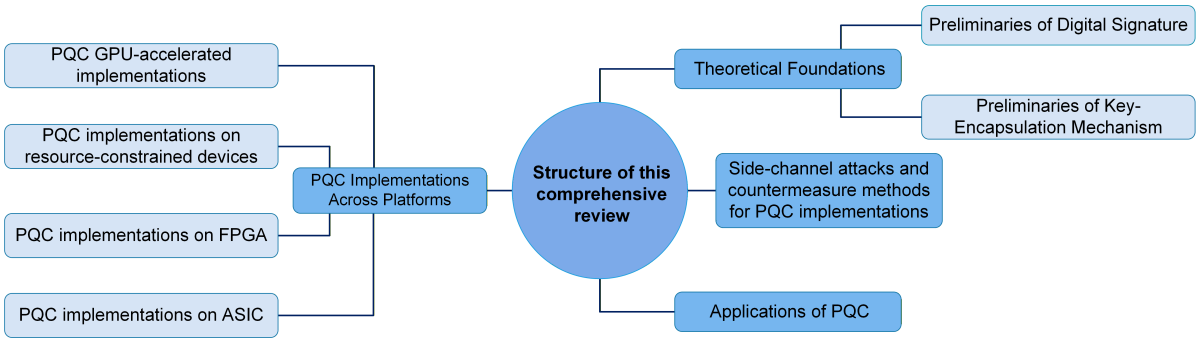

FIGURE 1. Organizational structure of the conceptual review.

them have already been published as Federal Information Processing Standards (FIPS): 203, 204, and 205 [3]. Specifically, CRYSTALS-Kyber was chosen as the primary cryptographic scheme for key establishment [4], and CRYSTALS-Dilithium for digital signatures [5], both based on module lattice problems. On the other hand, stateless practical hash-based incredibly nice cryptographic signature (SPHINCS$^+$), a hash-based scheme, was selected as a backup signature algorithm [6]. The two remaining algorithms are still in the process of standardization. The fast Fourier lattice-based compact signatures over NTRU (FALCON), a lattice-based scheme optimized for compact signatures and fast verification [7], and hamming quasi-cyclic (HQC), a code-based cryptosystem [8], have been selected for standardization as a backup to Kyber [9]. With the potential to serve as a foundational security layer, the new NIST cryptographic standards are expected to support a wide range of real-world applications, including internet banking, financial services, internet of things (IoT), and enterprise networks. Because they are designed to resist both quantum and classical attacks, these algorithms require substantially more computational resources than traditional symmetric and asymmetric cryptographic schemes. To meet these demands, optimized implementations have been proposed across platforms, including software, embedded systems, hardware, and software-hardware co-designs. These implementations are essential for achieving both performance and robustness in practical deployments.

Recognizing the importance and practical significance of PQC standards, several surveys summarized PQC algorithms and their theoretical foundations [10]–[14]. The works in [10], [11] presented overviews of the evolution of PQC throughout the three rounds of the NIST standardization process, outlined the theoretical underpinnings of PQC, and briefly introduced the round 3 finalist candidates. The study in [12] discussed the theoretical foundations of lattice-based cryptography (LBC), highlighted representative proposals from the NIST competition, and reviewed fundamental algorithms as well as cryptanalytic attacks against LBC schemes. More recently, the authors in [13] provided a comprehensive overview of the state of lattice-based PQC, with particular focus on number theoretic transform (NTT) implementations in lattice-based algorithms such as Kyber and Dilithium, which formed the core of the new PQC standards. Similarly, in [14], the authors introduced the mathematical hardness assumptions of the NTRU and learning with errors (LWE) problems and offered an extensive review of hardware implementations for LBC schemes. However, across these surveys, no work had yet provided a comprehensive review dedicated to the newly standardized NIST PQC finalists. Therefore, this survey fills that gap by offering a unified view of both the theoretical foundations and practical implementation aspects of the five newly standardized NIST PQC algorithms [9].

Specifically, we aim to provide an accessible overview of these schemes, rooted in diverse hard computational problems such as lattice-based, code-based, and stateless hash-based constructions, which often pose a steep learning curve for new researchers. We provide a concise introduction to the underlying concepts of each algorithm and explain their design principles as published in the NIST standardization reports. In addition, we summarize recent advances in hardware and software implementations of these algorithms, drawing from results published in leading journals and conferences. We aim to provide a comprehensive view of the current PQC implementation landscape, highlighting both efficient software solutions and hardware architectures designed to support the NIST PQC standards. These efficient implementations serve as valuable guidance for researchers and engineers working toward real-world deployment of PQC. We hope that this work offers researchers not only a clear understanding of the new NIST PQC standards but also an overview of the state-of-the-art implementation strategies across platforms. The main contributions of our paper are as follows:

1) We provide a concise and accessible introduction to the five NIST-selected PQC algorithms, highlighting their constructions, parameter sets, correctness, and functionality. This overview is designed to equip readers with the essential knowledge required to understand their role in the post-quantum era.
2) We deliver a comprehensive survey of recent software and hardware implementations, summarizing state-of-the-art proposals that optimize PQC schemes across diverse platforms. This conceptual review offers valuable

insights and serves as a practical guide for researchers entering the field.

3) We analyze hardware-level optimization strategies, discussing block-level design techniques and trade-offs between lightweight and high-performance architectures. By consolidating effective methods from recent works, we present a unified framework for optimizing PQC hardware implementations.

The structure of this conceptual review is divided into 3 main categories, as illustrated in Fig. 1. Section II introduces the theoretical foundations of standardized PQC algorithms, covering both digital signature algorithms and key encapsulation mechanisms. For each category, we highlight the mathematical assumptions and structural designs of the five algorithms that have been selected by NIST. Section III reviews recent implementation efforts across multiple platforms, including software, resource-constrained devices, and hardware architectures, while also discussing optimization strategies and summarizing existing implementations of the NIST PQC standards. Section IV introduces the vulnerabilities to side-channel attacks and the countermeasure methods for PQC implementations. Section V provides a brief overview of worldwide PQC applications, emphasizing the importance of these standards in practice. Finally, Section VI summarizes and concludes the paper.

II. THEORETICAL FOUNDATIONS

A. PRELIMINARIES OF DIGITAL SIGNATURE

Digital signature algorithm (DSA) is a cryptographic algorithm used to create digital signatures, which are represented in a computer as strings of bits, computed using a set of rules and parameters that enable verification of the signatory's identity and the integrity of the data. DSAs are built upon three fundamental components: key generation (KeyGen), signature generation (Sign), and signature verification (Verify). These core algorithms operate under well-defined parameter sets that specify values such as key sizes, hash lengths, and structural dimensions, allowing designers to balance performance, storage overhead, and security strength. A conceptual overview of the digital signature process is shown in Fig. 2. In typical operation, a user (the prover or signatory) generates, or obtains from a trusted authority, a public-private key pair. The private key must remain strictly confidential, while the public key may be freely distributed to any verifier. To sign a message, the signatory applies the private key to the message data, often after hashing it to a fixed-size digest, producing a digital signature. A verifier, using only the corresponding public key, can confirm whether the signature is valid for that message. This asymmetry ensures that only the legitimate private-key holder can produce valid signatures, while anyone with the public key can verify them.

However, the security of currently deployed digital signature standards, including RSA, the elliptic curve DSA and Edwards curve DSA [15], will be at risk if large-scale quantum computers are realized. After a long evaluation process and the elimination of numerous candidate algorithms, in September 2022, the status report on the third round of the NIST PQC standardization process announced Dilithium, FALCON and SPHINCS$^+$ as the selected algorithms. Dilithium was chosen as the primary standard for protecting digital signatures, renamed module-lattice-based DSA (ML-DSA), and formally specified in FIPS 204 (published in 2024). Similarly, SPHINCS$^+$ was selected as a backup in case ML-DSA proves vulnerable, renamed stateless hash-based DSA (SLH-DSA), and published in FIPS 205. Finally, FALCON was chosen as another backup standard, to be published in FIPS 206 and designated fast-Fourier transform (FFT) over NTRU-lattice-based DSA (FN-DSA) [9]. These post-quantum DSAs are designed to be strongly unforgeable, meaning that even after observing many valid message-signature pairs, an adversary cannot produce a new valid signature, whether on a previously signed message or a new one, without access to the private key. This ensures core security properties such as authenticity, integrity, and non-repudiation.

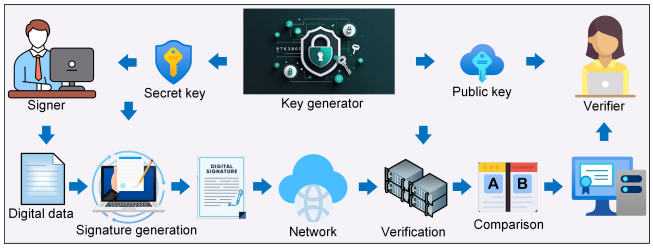

FIGURE 2. DSA process flow.

TABLE 1. Sizes (in bytes) of keys and signatures of DSA standards.

Sec.	ML-DSA			SLH-DSA			FALCON	
	2	3	5	1	3	5	1	5
sk	2560	4032	4896	64	96	128	7553	13953
pk	1312	1952	2592	32	48	64	897	1793
sig*	2420	3309	4627	7856/ 17088	16224/ 35664	29792/ 49856	666	1280

* Small signatures ('s')/ fast signature generation ('f').

1) ML-DSA Standard Overview and Explanation

The security of ML-DSA is based on the hardness of the module LWE (MLWE) problem and the short integer solution (SIS) problem [17]. It is believed to remain secure even against adversaries equipped with large-scale, fault-tolerant quantum computers [5]. To evaluate the computational complexity of this PQC algorithm, Fig.3 presents signature benchmarks on an x86-64 processor, using data collected by the Open Quantum Safe project—an open-source initiative that supports the transition to quantum-resistant cryptography [18] and is also referenced in the NIST report [3]. As shown in Table 1 and Fig. 3, ML-DSA offers relatively moderate key and signature sizes while achieving the fastest execution time among the three NIST-selected post-quantum DSA standards.

As with other asymmetric cryptographic schemes, ML-

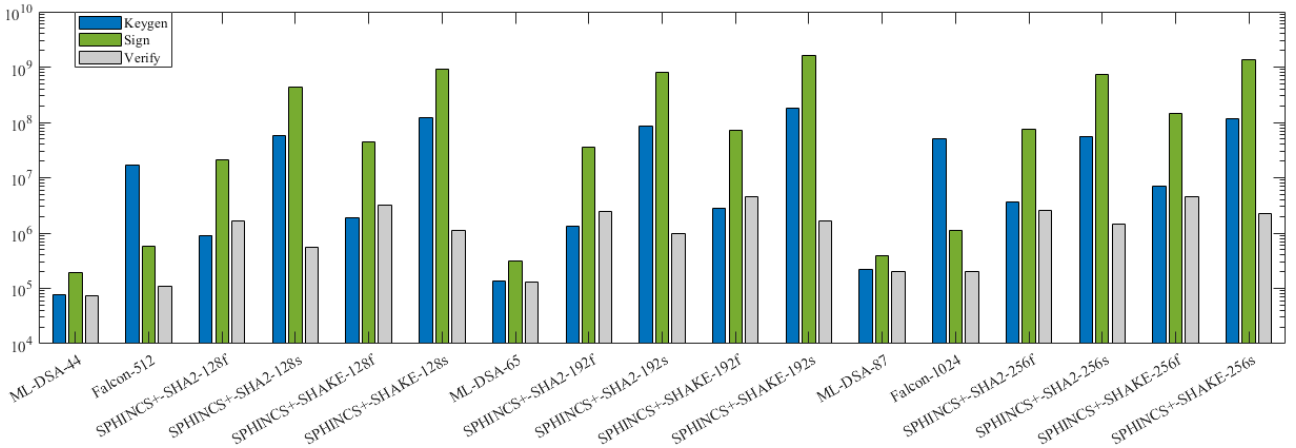

FIGURE 3. **DSA benchmarks on an x86-64 processor with AVX2 extensions (measured in clock cycles).**

DSA employs the Fiat-Shamir with Aborts construction [19] and consists of three main algorithms: ML-DSA.KeyGen (Algorithm 1), ML-DSA.Sign (Algorithm 2), and ML-DSA.Verify (Algorithm 3). The fundamental principle underlying ML-DSA, as well as similar lattice-based signature schemes, is to derive a signature scheme from an analogous interactive protocol. In this protocol, the prover possesses a matrix $\mathbf{A} \in R_q^{k\times\ell}$ and two secret vectors with small coefficients: $\mathbf{s}_1 \in R_q^{\ell}$ and $\mathbf{s}_2 \in R_q^{k}$. The prover then computes the public vector $\mathbf{t} \in R_q^{k}$ as $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$. Here, R denotes the ring of single-variable polynomials over $\mathbb{Z}$ modulo $X^{256}+1$. The vector $\mathbf{t}$ is compressed in the public key by discarding the d least significant bits of each coefficient, resulting in the polynomial vector $\mathbf{t}_1$. This vector, along with a public random seed ρ, forms the public key, which is transmitted to the verifier. The private key consists of the secret vectors $\mathbf{s}_1$ and $\mathbf{s}_2$, the discarded low-order bits of $\mathbf{t}$, a private random seed K, and a 64-byte hash tr of the public key.

During signing, the signer generates a vector $\mathbf{y} \in R_q^{\ell}$ based on the private randomness ρ'', which is derived as the digest of a hash function whose input is the concatenation of the private seed K, a random value rnd and 64-byte message representative μ. The random value rnd determines the signing mode: a default "hedged" variant that uses fresh randomness for each signature, or an optional "deterministic" variant that uses a fixed 32-byte zero string $\{0\}^{32}$. The signer then computes the commitment $\mathbf{w_1}$ and its hash $\tilde{c}$ (as shown in lines 9 and 10 of Algorithm 2). The hash $\tilde{c}$ is used to pseudo-randomly sample a challenge polynomial $c \in R_q^{\ell}$. Subsequently, the signer computes the response $\mathbf{z}$ and performs various validity checks; if any check fails, the rejection sampling loop repeats. Upon passing these checks, the signer computes a hint polynomial $\mathbf{h}$, which allows the verifier to reconstruct $\mathbf{w_1}$ exactly. Finally, the signature output consists of a byte encoding of $\tilde{c}$, $\mathbf{z}$ and $\mathbf{h}$.

With the public key and signature provided, any verifier can authenticate both the integrity of the message and the identity of the signer. The verification process begins by deriving the challenge polynomial c from the signer's commitment hash $\tilde{c}$. The verifier then computes an approximate value of the signer's vector $\mathbf{w}$ as $\mathbf{w'_{Approx}} = \mathbf{Az} - c\mathbf{t_1} \cdot 2^d$. Using the signer's hint $\mathbf{h}$, the verifier reconstructs $\mathbf{w'}_1$ from $\mathbf{w'_{Approx}}$. The hash image of message representative μ together with the reconstructed vector $\mathbf{w'}_1$ is then checked for consistency with the signer's original commitment hash $\tilde{c}$. The verifier also ensures that the signer's response $\mathbf{z}$ and hint $\mathbf{h}$ satisfy the scheme's validity constraints. If all these checks pass, the verifier concludes that the message is authentic and correctly bound to the signer's signature. This standard employs SHAKE256 and SHAKE128, two extendable-output hash functions (XOFs) from SHA-3 defined in FIPS 202 [20], for randomness generation and intermediate value sampling.

Algorithm 1 ML-DSA.KeyGen() [5]

Output: $pk = (\rho, \mathbf{t_1}), sk = (\rho, K, tr, \mathbf{s_1}, \mathbf{s_2}, \mathbf{t_0})$

1: $\xi \leftarrow \mathbb{B}^{32}$
2: $(\rho, \rho', K) \in \mathbb{B}^{32} \times \mathbb{B}^{64} \times \mathbb{B}^{32} \leftarrow \mathrm{H}(\xi \parallel k \parallel \ell)$
3: $\mathbf{A} \in R_q^{k\times\ell} \leftarrow \mathrm{ExpandA}(\rho)$
4: $(\mathbf{s_1}, \mathbf{s_2}) \in R_\eta^{\ell} \times R_\eta^{k} \leftarrow \mathrm{ExpandS}(\rho')$
5: $\mathbf{t} \leftarrow \mathrm{NTT}^{-1}(\hat{\mathbf{A}} \circ \mathrm{NTT}(\mathbf{s_1})) + \mathbf{s_2}$
6: $(\mathbf{t_1}, \mathbf{t_0}) \leftarrow \mathrm{Power2Round}(\mathbf{t})$
7: $tr \in \mathbb{B}^{64} \leftarrow \mathrm{H}(\rho \parallel \mathbf{t_1})$
8: **return** (pk, sk)

The correctness of the ML-DSA signature scheme can be formally established as follows. Given $\|c\mathbf{t_0}\|_\infty \geq \gamma_2$, Lemma 1 in the Dilithium specification [21] guarantees that:

1) $\mathrm{UseHint}_q(\mathbf{h}, \mathbf{w}\text{-}c\mathbf{s_2}+c\mathbf{t_0}, 2\gamma_2) = \mathrm{HighBits}_q(\mathbf{w}\text{-}c\mathbf{s_2}, 2\gamma_2)$.
2) Since $\mathbf{w}\text{-}c\mathbf{s_2} = \mathbf{Ay}\text{-}c\mathbf{s_2} = \mathbf{Az}\text{-}c\mathbf{t}$ and $\mathbf{w}\text{-}c\mathbf{s_2}+c\mathbf{t_0} = \mathbf{Az}\text{-}c\mathbf{t_1} \cdot 2^d$, the verifier computes:

$$\begin{aligned}&\mathrm{UseHint}_q(\mathbf{h}, \mathbf{A}z - c\mathbf{t_1} \cdot 2^d, 2\gamma_2)\\ &= \mathrm{HighBits}_q(\mathbf{w} - c\mathbf{s_2}, 2\gamma_2)\end{aligned} \tag{1}$$

3) Moreover, because β is chosen such that $\|c\mathbf{s_2}\|_\infty \leq \beta$ and the signing condition ensures that $\mathrm{LowBits}_q(\mathbf{w}\text{-}c\mathbf{s_2}, 2\gamma_2) < \gamma_2 - \beta$, Lemma 2 in [21] implies:

$$\begin{aligned}&\mathrm{HighBits}_q(\mathbf{w} - c\mathbf{s_2}, 2\gamma_2)\\ &= \mathrm{HighBits}_q(\mathbf{w} - c\mathbf{s_2} + c\mathbf{s_2}, 2\gamma_2) = \mathbf{w_1}\end{aligned} \tag{2}$$

Algorithm 2 ML-DSA.Sign(sk, M, ctx) [5]

Input: $sk = (\rho, K, tr, \mathbf{s_1}, \mathbf{s_2}, \mathbf{t_0}), M \in \{0,1\}^*, ctx$
Output: $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$

$M' \leftarrow$ (IntToBytes(0,1) $\|$ IntToBytes($|ctx|$,1) $\|$ ctx) $\|$ M
$\hat{\mathbf{A}} \in R_q^{k \times \ell} \leftarrow$ ExpandA(ρ)
$\mu \leftarrow$ H($tr \parallel M'$), $rnd \leftarrow \mathbb{B}^{32}$ *or* $\{0\}^{32}$
$\rho'' \leftarrow$ H($K \parallel rnd \parallel \mu$)
$\kappa \leftarrow 0$, $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
while $(\mathbf{z}, \mathbf{h}) = \perp$ **do**
  $\mathbf{y} \in R_q^{\ell} \leftarrow$ ExpandMask(ρ'', κ)
  $\mathbf{w} \leftarrow \text{NTT}^{-1}(\hat{\mathbf{A}} \circ \text{NTT}(\mathbf{y}))$
  $\mathbf{w_1} \leftarrow$ HighBits($\mathbf{w}$)
  $\tilde{c} \leftarrow$ H($\mu \parallel \mathbf{w_1}$) $\leftarrow \mathbb{B}^{\lambda/4}$
  $c \in R_q \leftarrow$ SampleInBall($\tilde{c}$)
  $\mathbf{z} \leftarrow \mathbf{y} + c\mathbf{s_1}$
  $\mathbf{r_0} \leftarrow$ LowBits($\mathbf{w}$ - $c\mathbf{s_2}$, $2\gamma_2$)
  if $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ **or** $\|\mathbf{r_0}\|_\infty \geq \gamma_2 - \beta$ **then**
    $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
  else
    $\mathbf{h} \leftarrow \text{MakeHint}(-c\mathbf{t_0}, \mathbf{w} - c\mathbf{s_2} + c\mathbf{t_0})$
    if $\|c\mathbf{t_0}\|_\infty \geq \gamma_2$ **or** $\sum \mathbf{h_i} > \omega$ **then**
      $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
    end if
  end if
  $\kappa \leftarrow \kappa + \ell$
end while
return $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$

Algorithm 3 ML-DSA.Verify(pk, M, σ, ctx) [5]

Input: $pk = (\rho, \mathbf{t_1}), M \in \{0,1\}^*, \sigma = (\tilde{c}, \mathbf{z}, \mathbf{h}), ctx$
Output: Valid or Invalid

$M' \leftarrow (0 \parallel |ctx| \parallel ctx) \parallel M$
$\hat{\mathbf{A}} \in R_q^{k \times \ell} \leftarrow$ ExpandA(ρ)
$\mu \leftarrow$ H(H($\rho \parallel \mathbf{t_1}$) $\parallel M'$)
$c \in R_q \leftarrow$ SampleInBall($\tilde{c}$)
$\mathbf{w_1'} \leftarrow$ UseHint($\mathbf{h}$, $\mathbf{Az} - c\mathbf{t_1} \cdot 2^d$)
if $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$ & $\tilde{c} =$ H($\mu \parallel \mathbf{w_1'}$) **then**
  return Valid
end if
return Invalid

Consequently, the commitment $\mathbf{w_1}$ generated by the signer matches the commitment $\mathbf{w_1}'$ recovered by the verifier. This ensures that the commitment hash $\tilde{c}$ in both procedures is equal, which completes the verification process and guarantees correctness of the ML-DSA signature scheme.

Based on its module-lattice-based structure, ML-DSA can readily adjust its security strength through different parameter sets. NIST has proposed three ML-DSA parameter sets, each designed to meet specific security strength categories defined in its Submission Requirements and Evaluation Criteria [2]. The parameter set names follow the format ML-DSA-$k\ell$, where (k, ℓ) represent the dimensions of the public matrix $\mathbf{A}$, which directly determine the targeted security strength category. Accordingly, ML-DSA-44 is specified to achieve security strength category 2, ML-DSA-65 targets category 3, and ML-DSA-87 corresponds to category 5.

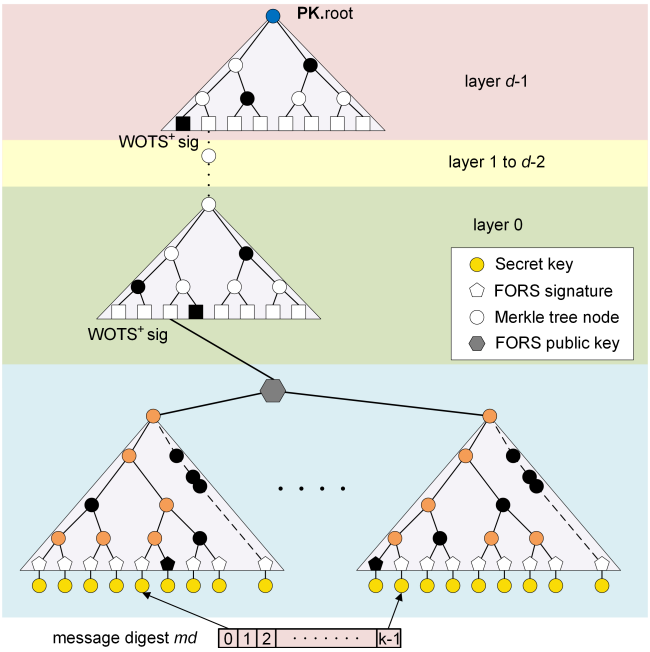

FIGURE 4. An overall construction of SLH-DSA signature.

2) Stateless Hash-Based Digital Signature Standard Overview and Explanation

SLH-DSA is a stateless hash-based digital signature scheme standardized in FIPS 205 [6], constructed from the composition of two nested hash-based primitives: a few-time signature scheme, the forest of random subsets (FORS), and a multi-time signature scheme, the eXtended Merkle signature scheme (XMSS). XMSS itself is built using the hash-based Winternitz one-time signature plus (WOTS$^+$) scheme as a core component. SLH-DSA is derived from the selected PQC algorithm SPHINCS$^+$, whose security can be formally verified based on the security properties of its underlying hash functions [22]. The security of SLH-DSA relies on the presumed difficulty of finding preimages for the hash functions, along with several related properties of these functions. The scheme benefits from very short public and private keys, as shown in Table 1; however, it produces comparatively long signatures and exhibits a relatively slow signing process. Benchmarking on an x86-64 processor [18] (Fig. 3) shows that the sign and verify times of SPHINCS$^+$ are significantly higher than those of ML-DSA, highlighting the inherent computational complexity of hash-based cryptographic schemes when implemented in software.

SLH-DSA was selected for standardization because it offers a practical—though large and slow—signature scheme with security that appears robust and is based on a fundamentally different set of assumptions from those of the other signature schemes selected for standardization [3]. It supports 12 parameter sets, divided into three NIST-defined security strength categories. Each parameter set determines a trade-off between the computational complexity of the signing and verification processes and the resulting signature size. The

TABLE 2. Initial function component of SLH-DSA.

Name	Function	Input	Output
wots_pkGen	Generates a WOTS$^+$ public key	**SK**.seed, **PK**.seed, **ADRS**	WOTS$^+$ pk
wots_sign	Generates a WOTS$^+$ *sig* on n-byte message	M, **SK**.seed, **PK**.seed, **ADRS**	WOTS$^+$ *sig*
wots_pkFromSig	Computes WOTS$^+$ pk from message and its signature	*sig*, M, **PK**.seed, **ADRS**	WOTS$^+$ pk_{sig}
xmss_node	Computes the root of a Merkle subtree of WOTS$^+$	**SK**.seed, index i, height z, **PK**.seed, **ADRS**	root *node*
xmss_sign	Generates an XMSS signature	M, **SK**.seed, index idx, **PK**.seed, **ADRS**	SIG_{XMSS}
xmss_pkFromSig	Computes an XMSS public key from XMSS signature	idx, SIG_{XMSS}, M, **PK**.seed, **ADRS**	root $node[0]$
ht_sign	Generates a hypertree signature	M, **SK**.seed, **PK**.seed, idx_{tree}, idx_{leaf}	SIG_{HT}
ht_verify	Verifies a hypertree signature	M, SIG_{HT}, **PK**.seed, idx_{tree}, idx_{leaf}, **PK**.root	Boolean
fors_skGen	Generates FORS private-key value	**SK**.seed, **PK**.seed, **ADRS**, idx	FORS private-key value
fors_node	Computes the root of a Merkle subtree of FORS public values	**SK**.seed, index i, height z, **PK**.seed, **ADRS**	root *node*
fors_sign	Generates a FORS signature	digest md, **SK**.seed, **PK**.seed, **ADRS**	SIG_{FORS}
fors_pkFromSig	Computes a FORS public key from FORS signature	SIG_{FORS}, md, **PK**.seed, **ADRS**	FORS public key

parameter set names indicate the hash function family used (SHA-2 [23] or SHAKE [20]), the length in bits of the security parameter n, and whether the set was designed for relatively small signatures ('s') or for relatively fast signing ('f').

Conceptually, the SLH-DSA signing process is illustrated in Fig. 4. At the lowest level, the FORS method signs the original input message. However, since authenticating a FORS public key requires a large number of WOTS$^+$ keys, directly generating all the necessary authentication paths would be computationally expensive. To address this scalability issue, SLH-DSA employs a hypertree structure—an arrangement of multiple XMSS trees stacked hierarchically—which significantly reduces the number of leaf computations from 2^h to $2^{h'}$ per XMSS subtree. Accordingly, the hypertree has an overall height h, divided into d layers of XMSS subtrees, each of height h' ($h = d \cdot h'$). At the top layer, there is a single XMSS tree whose root forms the public key of the scheme. The next layer contains $2^{h'}$ XMSS trees, and this pattern continues downward until the lowest layer, which contains $2^{(d-1)h'}$ XMSS trees. Each XMSS tree authenticates WOTS$^+$ public keys, which in turn authenticate either FORS public keys (in the lowest layer) or the roots of XMSS trees from the layer beneath (in higher layers). At the lowest layer, the WOTS$^+$ keys are directly used to sign 2^h FORS public keys. The hypertree signature—composed of a chain of XMSS and WOTS$^+$ signatures across all layers—thus provides complete authentication for the FORS root included in the SLH-DSA signature. The main SLH-DSA components and their respective functions are as follows (also summarized in Table 2):

FORS: A few-time signature scheme used to sign message digests. FORS splits the message digest into multiple segments, each used to select a subset of secret keys from which a public key component is computed. The concatenated results are then hashed to produce the FORS public key. FORS is secure for only a small, bounded number of signatures per key pair; hence, it is embedded inside XMSS for long-term use. In SLH-DSA, the FORS root is authenticated via the hypertree.

Hypertree (HT): A hierarchy of XMSS trees arranged in d layers. The lowest layer's XMSS keys sign the FORS public keys, while higher-layer XMSS keys authenticate the XMSS public keys in the layer below. This recursive authentication structure enables very large signing capacities without regenerating all WOTS$^+$ keys upfront.

XMSS: A stateful hash-based signature scheme that combines WOTS$^+$ with Merkle trees. An XMSS tree of height h' can authenticate $2^{h'}$ WOTS+ public keys using a single n-byte root. Each WOTS$^+$ key can only sign one message, so the XMSS layer provides multi-time signing capability while retaining compact public keys.

WOTS$^+$: The core one-time signature primitive used within both XMSS and the hypertree. In SLH-DSA, WOTS$^+$ signs the FORS root in the lowest layer, or the XMSS subtree roots in higher layers. The WOTS$^+$ public key is placed as a leaf in a Merkle tree, and its validity is established via an authentication path up to the subtree root. This process is repeated layer-by-layer until the top-level root is reached.

Address (ADRS): A 32-byte structured field included in every hash computation within SLH-DSA. ADRS encodes contextual information such as the tree index, leaf index, and key type, serving as a domain-separation and tweaking mechanism to ensure that all hash invocations are uniquely bound to their position in the signature hierarchy.

Similar to other asymmetric digital signature schemes, SLH-DSA consists of three main algorithms: key generation (Algorithm 4), signature generation (Algorithm 5) and verification (Algorithm 6). As the algorithm names suggest, slh_keygen is responsible for generating an SLH-DSA key pair. The security of SLH-DSA relies on the presumed difficulty of finding preimages for cryptographic hash functions. Its operation can therefore be described through the procedures of its main algorithms. The SLH-DSA public key contains an n-byte public seed **PK**.seed and **PK**.root, which is the root of the top-layer XMSS tree. The private key contains two random and secret n-byte values: **SK**.seed, used to generate all WOTS$^+$ and FORS private key elements, and **SK**.prf, used to generate the message randomization value in the randomized hashing process. In Keygen, **PK**.seed, **SK**.seed, and **SK**.prf shall be generated using an approved random bit

Algorithm 4 slh_keygen() [6]

Output: SLH_DSA key pair (SK,PK).

1: **SK**.seed, **SK**.prf, **PK**.seed $\xleftarrow{\$} \mathbb{B}^n \times \mathbb{B}^n \times \mathbb{B}^n$
2: **if** (**SK**.seed || **SK**.prf || **PK**.seed == NULL) **then**
3: **return** $\perp$
4: **ADRS** $\leftarrow$ toByte(0, 32)
5: **ADRS**.setLayerAddress(d-1)
6: **PK**.root $\leftarrow$ xmss_node(**SK**.seed, 0, h', **PK**.seed, **ADRS**)
7: **return** ((**SK**.seed, **SK**.prf, **PK**.seed, **PK**.root), (**PK**.seed, **PK**.root))

generator [24], where the generator instantiation must support at least $8n$ bits of security strength. The value **PK**.root is then computed as the root of the top-level XMSS tree.

The slh_sign algorithm begins by generating m-byte message digest, computed as the hash output of the message randomizer concatenated with the message itself. This digest is then partitioned into components used to determine the signing parameters: one part is signed using the FORS key, another part selects the XMSS tree index (idx_{tree}) and a third part selects the WOTS$^+$ key and the corresponding FORS key within that XMSS tree (idx_{leaf}). The FORS signature (SIG_{FORS}) is then computed, and the corresponding FORS public key (PK_{FORS}) is derived. Following this, the hypertree signature is generated by signing PK_{FORS} using the WOTS$^+$ and XMSS structure. SLH-DSA, like ML-DSA, supports both "hedged" and "deterministic" operational variants, distinguished by the method used to obtain the additional randomness value *addrnd*. In the default hedged variant, *addrnd* is generated using an approved random bit generator, while in the deterministic variant, *addrnd* is replaced by **PK**.seed.

The slh_verify algorithm first ensures that the signature length matches the expected value. It then recomputes the message digest using the message and the message randomizer extracted from the signature. The same FORS procedures as in slh_sign are executed to reconstruct PK_{FORS}, after which hypertree verification (ht_verify) is performed. This process recomputes the XMSS public keys from the supplied XMSS/WOTS$^+$ signatures and compares the result with **PK**.root. If the verification succeeds, the correct FORS public key has been recovered, and the signature is confirmed to be valid for the given message and public key.

As a hash-based signature scheme, the correctness of SLH-DSA stems entirely from the construction of its building blocks—WOTS$^+$, XMSS, and the Hypertree (Table 2)—each of which plays a distinct role in signing and verification. In WOTS$^+$, the signing algorithm takes secret key elements and repeatedly hashes them to generate a one-time signature for the message digest, while the verification algorithm reapplies the same hash function to the signature parts to reconstruct the public key. Because hashing is deterministic, the reconstructed public key always equals the stored value when the correct secret key is used. XMSS and the Hypertree extend this process by selecting a WOTS$^+$ keypair leaf and attaching an authentication path consisting of sibling nodes in the Merkle tree; the verifier then recomputes the root

Algorithm 5 slh_sign(M, ctx, SK) [6]

Input: Message M, context string ctx, private key SK.
Output: SLH_DSA signature SIG.

1: $addrnd \xleftarrow{\$} \mathbb{B}^n$
2: $M' \leftarrow$ toByte(0, 1) || toByte(|ctx|, 1) || ctx || M
3: **ADRS** $\leftarrow$ toByte(0, 32)
4: $opt\_rand \leftarrow addrnd$ (**PK**.seed for deterministic variant)
5: SIG $\leftarrow R \leftarrow \mathbf{PRF}_{msg}$(**SK**.prf, opt_rand, M)
6: $digest \leftarrow \mathbf{H}_{msg}(R$, **PK**.seed, **PK**.root, $M)$
7: md, idx_{tree}, $idx_{leaf} \leftarrow digest$
8: **ADRS**.setTreeAddress(idx_{tree})
9: **ADRS**.setTypeAndClear(FORS_{tree})
10: **ADRS**.setKeyPairAddress(idx_{leaf})
11: $\text{SIG}_{FORS} \leftarrow$ fors_sign(md, **SK**.seed, **PK**.seed, **ADRS**)
12: SIG $\leftarrow$ SIG || SIG_{FORS}
13: $\text{PK}_{FORS} \leftarrow$ fors_pkFromSig(SIG_{FORS}, md, **PK**.seed, **ADRS**)
14: $\text{SIG}_{HT} \leftarrow$ ht_sign($\mathbf{PK}_{FORS}$, **SK**.seed, **PK**.seed, idx_{tree}, idx_{leaf})
15: **return** SIG $\leftarrow$ SIG || SIG_{HT}

Algorithm 6 slh_verify(M, SIG, ctx, PK) [6]

Input: Message M, signature SIG, context string ctx, PK.
Output: Boolean.

1: $M' \leftarrow$ toByte(0, 1) || toByte(|ctx|, 1) || ctx || M
2: **if** $|\text{SIG}| \neq (1 + k(1 + a) + h + d \cdot len) \cdot n)$ **then**
3: **return** false
4: $R \leftarrow \text{SIG}[0 : n]$
5: $\text{SIG}_{FORS} \leftarrow \text{SIG}[n : (1 + k(1 + a)) \cdot n]$
6: $\text{SIG}_{HT} \leftarrow \text{SIG}[(1+k(1+a)) \cdot n : (1+k(1+a)+h+d \cdot len) \cdot n]$
7: $digest \leftarrow \mathbf{H}_{msg}(R$, **PK**.seed, **PK**.root, $M)$
8: md, idx_{tree}, $idx_{leaf} \leftarrow digest$
9: **ADRS**.setTreeAddress(idx_{tree})
10: **ADRS**.setTypeAndClear(FORS_{tree})
11: **ADRS**.setKeyPairAddress(idx_{leaf})
12: **return** ht_verify(PK_{FORS}, SIG_{HT}, **PK**.seed, idx_{tree}, idx_{leaf}, **PK**.root)

deterministically by hashing along this path. If the signer has provided the correct leaf, the recomputed root equals the public key. As detailed in slh_sign (Algorithm 5) and slh_verify (Algorithm 6), the message digest, tree index and leaf index are consistently derived from the same hash function applied to the input message. Thus, as long as the message is unmodified, the verification procedure will always reproduce the same root and confirm that the signature indeed corresponds to the original signed message.

3) FALCON Overview and Explanation

FALCON is a lattice-based signature scheme that adopts "hash-and-sign" paradigm [7]. Its theoretical security is grounded in proof of unforgeability in the quantum-accessible random oracle model, relying on the hardness of the SIS problem over NTRU lattices [25]. At a high level, FALCON follows the Gentry-Peikert-Vaikuntanathan (GPV) framework [26] for constructing hash-and-sign schemes from lattice-based trapdoor functions with preimage sampling. The trapdoor preimage sampling algorithm in FALCON is notably intricate, requiring the use of floating-point arithmetic and sophisticated data structures, such as the FALCON tree, which makes this scheme considerably challenging to implement in practice [3]. As

Algorithm 7 FALCON.KeyGen(ϕ, q) [7]

Require: A monic polynomial $\phi \in \mathbb{Z}[x]$, a modulus q.
Ensure: A secret key sk and public key pk.

$f, g, F, G \leftarrow$ NTRUGen(ϕ, q)
$\mathbf{B} \leftarrow \left[\begin{array}{c|c} g & -f \\ \hline G & -F \end{array}\right]$
$\hat{\mathbf{B}} \leftarrow$ FFT($\mathbf{B}$)
$\mathbf{G} \leftarrow \hat{\mathbf{B}} \times \hat{\mathbf{B}}^{\star}$
$\mathbf{T} \leftarrow$ ffLDL*($\mathbf{G}$)
for each leaf `leaf` of $\mathbf{T}$
  leaf.value $\leftarrow \sigma/\sqrt{\text{leaf.value}}$
sk $\leftarrow (\hat{\mathbf{B}}, \mathbf{T})$
pk $\leftarrow h \leftarrow gf^{-1} \mod q$
return pk, sk

Algorithm 8 FALCON.Sign(m, sk, $\lfloor\beta^2\rceil$) [7]

Require: A message m, secret key sk, a bound $\lfloor\beta^2\rfloor$.
Ensure: A signature sig of m.

$\mathbf{r} \leftarrow \{0,1\}^{320}$
$c \leftarrow$ HashToPoint(r‖m, q, n)
$\mathbf{t} \leftarrow (\frac{1}{q}\text{FFT}(c) \odot \text{FFT}(F), \frac{1}{q}\text{FFT}(c) \odot \text{FFT}(f))$
do
  do
    $\mathbf{z} \leftarrow$ ffSampling$_n$($\mathbf{t}$, $\mathbf{T}$)
    $\mathbf{s}$ = ($\mathbf{t}$ - $\mathbf{z}$)$\hat{\mathbf{B}}$
  while $\| \mathbf{s} \|^2 > \lfloor\beta^2\rfloor$
  $(s_1, s_2) \leftarrow$ invFFT($\mathbf{s}$)
  s $\leftarrow$ Compress(s_2, 8 · sbytelen - 328)
while s = $\perp$ **return** sig = ($\mathbf{r}$, s)

Algorithm 9 FALCON.Verify(m, sig, pk, $\lfloor\beta^2\rceil$) [7]

Require: A message m, signature sig = ($\mathbf{r}$, $\mathbf{s}$), a bound $\lfloor\beta^2\rfloor$, public key pk = $h \in \mathbb{Z}_q[x]/(\phi)$.
Ensure: Accept or reject.

$c \leftarrow$ HashToPoint(r‖m, q, n)
$s_2 \leftarrow$ Decompress($\mathbf{s}$, 8 · sbytelen - 328)
if ($s_2 = \perp$) then
  reject
$s_1 \leftarrow c$ - $s_2 h \mod q$
if $\|(s_1, s_2)\|^2 \leq \lfloor\beta^2\rfloor$ then
  accept
else
  reject

shown in Table 2 and Fig. 3, FALCON offers very fast signature verification; however, its signing and key generation operations are significantly slower compared to ML-DSA. On the other hand, FALCON achieves the smallest bandwidth (combined size of public key and signature), which can make it an attractive choice in bandwidth-constrained protocol scenarios and certain specialized applications. Owing to these characteristics, FALCON was selected for standardization in 2022 [3] as a backup option to ML-DSA. This algorithm remains under development and is expected to be officially released as FIPS 206 under the name FN-DSA.

Unlike other NIST submissions that propose a wide range of parameter sets, FALCON specifies only two, corresponding to NIST security levels I and V. The parameters are determined primarily by the degree n of the underlying NTRU polynomial ring. At a high level, FALCON consists of three algorithms common to digital signature schemes: key generation (Algorithm 7), signature generation (Algorithm 8), and signature verification (Algorithm 9). The FALCON key generation process begins with solving the NTRU equation, which requires finding polynomials f, g, F, $G \in \mathbb{Z}[x]/(\phi)$, where $\phi = x^n + 1$ is monic and irreducible polynomial, such that the relation: fG - $gF = q \mod \phi$ holds. The polynomials f and g are sampled at random using a discrete Gaussian distribution to ensure cryptographic security. Once these are obtained, a recursive tower-of-rings method is applied to determine the matching polynomials F and G [7]. With suitable values of f, g, F, G computed, they are transformed into a trapdoor structure known as the FALCON tree $\mathbf{T}$. This is achieved by performing an LDL* decomposition on the Gram matrix $\mathbf{G}$, where $\mathbf{G} = \mathbf{LDL}^*$. In this, $\mathbf{L}$ is a lower triangular matrix with ones along the diagonal, and $\mathbf{D}$ is diagonal matrix:

$$\mathbf{L} = \left[\begin{array}{c|c} 1 & 0 \\ \hline L_{10} & 1 \end{array}\right] \text{ and } \mathbf{D} = \left[\begin{array}{c|c} D_{00} & 0 \\ \hline 0 & D_{11} \end{array}\right] \quad (3)$$

The root value L_{10} is stored as the top node of the FALCON tree, after which the diagonal entries of $\mathbf{D}$ are recursively split into submatrices. Each submatrix is decomposed in the same manner, building the tree until reaching the base field $\mathbb{Q}$. The private key consists of the FFT representation of a wrapped matrix $\hat{\mathbf{B}}$ together with trapdoor tree $\mathbf{T}$, while public key is given by $h = gf^{-1} \mod q$.

In the signing phase, signer first computes a hash value $c \in \mathbb{Z}_q[x]/(\phi)$ from random salt r and message m. A preimage vector $\mathbf{t}$ corresponding to c is then determined. To generate the signature without leaking the secret key, FALCON employs a fast Fourier sampling method, which is a recursive discrete Gaussian trapdoor sampler operating over the precomputed FALCON tree $\mathbf{T}$, derived from the private key. This algorithm produces a short vector $\mathbf{z}$ such that rounding ($\mathbf{t}$ - $\mathbf{z}$)$\hat{\mathbf{B}}$ yields a valid signature vector $\mathbf{s} = (s_1, s_2)$ that satisfies the equation $s_1 + s_2 h = c \mod q$. Since many solutions exist, FALCON ensures security by sampling only short vectors. Therefore, if the resulting vector does not satisfy the predefined norm bound β, rejection sampling is performed until a valid signature is obtained. The final signature consists of the salt r and a compressed form of $\mathbf{s}$, where only s_2 is stored explicitly.

For verification, the verifier receives the public key, message, and signature, and recomputes c by hashing the salt and message, mirroring the signing process. The polynomial s_2 is recovered from the compressed signature, and s_1 is reconstructed as $c - s_2 h \mod q$, normalized to the symmetric range from $\lceil -q/2 \rceil$ to $\lfloor q/2 \rfloor$. The signature is accepted if and only if the squared Euclidean norm of the pair (s_1, s_2) is less than the threshold $\lfloor\beta^2\rfloor$, thereby ensuring both correctness and adherence to the security requirements.

To illustrate the step-by-step operations of FALCON in a simplified setting, we present a toy example with parameters n=4 and q=17. First, we generate two small polynomials f,

$g \in \mathbb{Z}_q[x]/(\phi)$, and then solve the NTRU equation to obtain corresponding polynomials F, G. For this example, we select $f = [1,0,0,0](= 1)$ and $g = [1,-1,0,0](= 1 - x)$. Solving the NTRU equation yields $F = [3,0,4,8]$ and $G = [12,-3,4,4]$, which satisfy $fG - gF = [17,0,0,0]$. The public key polynomial h is derived from the relation $h = g \cdot f^{-1} \bmod q = [1,16,0,0]$.

During signature generation, a challenge polynomial $c \in \mathbb{Z}_q[x]/(\phi)$, derived from the hash of the message m and a random salt, is used to represent the verification target. In this example, we take $c = [16,0,3,2]$. The goal is to compute two short polynomials $s_1, s_2 \in \mathbb{Z}[x]/(\phi)$ such that they satisfy $s_1 + s_2 h = c \bmod q$. One possible solution is $s_1 = [1,0,2,1]$ and $s_2 = [1,1,0,1]$, which indeed satisfies the equation modulo q. However, the essential security of FALCON is based on the SIS problem, which requires that the produced signature $\mathbf{s}$ must not only satisfy the equation but also be bounded in norm. Without this bound, trivial or excessively large solutions could always be found, undermining both correctness and security. To enforce this, FALCON introduces a rejection sampling step: the private polynomials f, g, F, G are sampled from a discrete Gaussian distribution with standard deviation δ and the resulting signature vector is only accepted if $\|\mathbf{s}\|$ is smaller than a predefined threshold β. This norm bound ensures two critical properties: (*i*) it guarantees that signatures are "short" enough to prevent leakage of secret key information through statistical deviations, and (*ii*) it ties the signature problem to the SIS assumption, since finding a valid short solution to the linear relation is computationally hard for an attacker without knowledge of the secret key.

In the verification process, the verifier only requires the public key h, the received challenge polynomial c and the signature component s_2. Using these values, the verifier reconstructs $s_1 \equiv c - s_2 h \bmod q$ and normalizes its coefficients into the range (-q/2, q/2). The verifier then checks whether the Euclidean norm of (s_1, s_2) is within the maximum allowed signature bound β. If this bound is satisfied, the signature is accepted as valid; otherwise, it is rejected. Consequently, any modification of the message m leads to a different challenge c, which produces a distinct s_1 that fails the norm check. This mechanism guarantees that only signatures generated from the true message are accepted, thereby ensuring both correctness and security.

B. KEY-ENCAPSULATION MECHANISM PRELIMINARIES

Key-encapsulation mechanism (KEM) is a fundamental cryptographic tool designed to enable two parties to securely agree upon a shared secret key across an insecure communication channel. This shared secret is subsequently used as a symmetric key to facilitate various cryptographic functions such as encryption and authentication, ensuring confidentiality and integrity in communication systems [27]. Current key-establishment protocols, including those outlined in standards like SP 800-56A and 800-56B [28], [29], rely on mathematical problems that are vulnerable to attacks by sufficiently powerful quantum computers. The advent of quantum computing thus threatens the security guarantees of these classical schemes. To address this emerging risk, NIST launched PQC standardization initiative aimed at identifying and validating quantum-resistant algorithms suitable for widespread adoption. Following comprehensive evaluations spanning multiple rounds, the lattice-based cryptosystem Kyber was selected as the leading candidate for key encapsulation. This algorithm, standardized as ML-KEM, offers strong security assurances based on the hardness of lattice problems, which remain intractable even in the presence of quantum adversaries. Its standardization, finalized in 2024, marks a significant milestone in securing future communication infrastructures. Additionally, more recent developments have recognized HQC, a KEM based on quasi-cyclic codes, as a complementary candidate, expanding the diversity of post-quantum solutions.

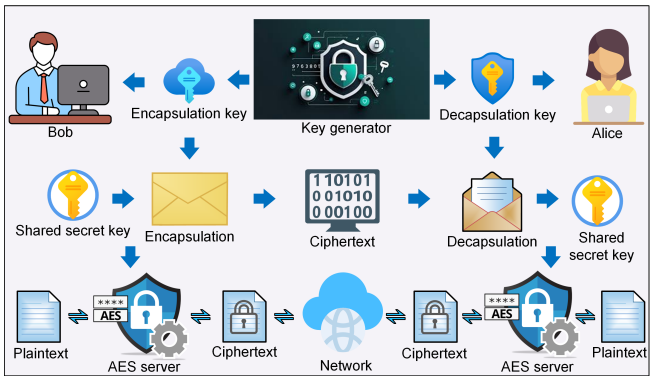

FIGURE 5. Overall view of key establishment using a KEM.

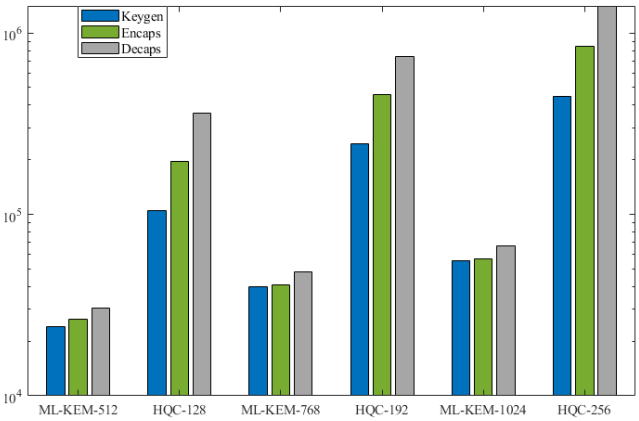

FIGURE 6. KEM Benchmarks on x86-64 processor with AVX2 extensions.

From a conceptual perspective, KEM is typically specified by three fundamental algorithms: Key generation, encapsulation (Encaps), and decapsulation (Decaps). The KeyGen algorithm produces a key pair consisting of a public encapsulation key (ek) and a private decapsulation key (dk). Using ek, the Encaps algorithm generates both a ciphertext and a shared secret key K, while the Decaps algorithm, applied to the ciphertext and dk, recovers the same shared secret key. To address varying performance and security requirements,

TABLE 3. Sizes (in bytes) of keys and ciphertext of KEM standards.

Parameters	ML-KEM			HQC		
	1	3	5	1	3	5
Encapsulation key	800	1184	1568	2241	4514	7237
Decapsulation key	1632	2400	3168	2321	4602	7333
Ciphertext	768	1088	1568	4433	8978	14421
Shared secret key	32	32	32	32	32	32

the NIST PQC standardization effort specifies three security categories for KEM schemes, enabling a trade-off between execution time and cryptographic strength. Parameter sets corresponding to each security category allow implementers to fine-tune bandwidth, latency, and computational cost for different application domains.

In a typical key exchange scenario as described in Fig. 5, Alice either generates her own key pair by running KeyGen or securely obtains it from a trusted authority to ensure no leakage of sensitive information. She then sends the public ek to Bob. Upon receiving Alice's encapsulation key, Bob executes Encaps, producing his copy K of the shared secret key along with an associated ciphertext. Bob transmits the ciphertext to Alice, who applies Decaps with her private decapsulation key to recover her copy K' of the shared secret key. The correctness property of KEMs ensures that, under normal operation, $K' = K$. This key, indistinguishable from a uniformly random string to any external adversary, is then used as the symmetric key for efficient cryptographic schemes such as the advanced encryption standard (AES) [30]. Leveraging AES in this way combines the strong security guarantees of post-quantum key establishment with the high performance of symmetric encryption, ensuring rapid encryption and decryption while maintaining robust protection. This mechanism is common to both ML-KEM and HQC, where the shared secret key established through this process is considered secure and can be implemented in software, firmware, hardware, or any combination thereof. The key and ciphertext sizes (in bytes) for standardized PQC KEMs are summarized in Table 2, while execution times, measured on an x86-64 platform with AVX2 optimizations [18], are illustrated in Fig. 6. Experimental results demonstrate that ML-KEM, a primary standard, offers notable advantages in both bandwidth efficiency and computational performance, achieving faster execution across all core procedures compared to alternative schemes.

1) Module-Lattice-Based KEM Overview and Explanation

ML-KEM is a PQC key-encapsulation mechanism based on Kyber, a lattice-based cryptographic primitive designed for secure key exchange over public, potentially adversarial, communication channels [31]. Unlike conventional key establishment schemes described in NIST SP 800-56A and SP 800-56B, which rely on the hardness assumptions of integer factorization or the discrete logarithm problem, ML-KEM is designed to resist attacks from large-scale fault-tolerant quantum computers. This security is grounded in the

Algorithm 10 ML-KEM.KeyGen() [4]

Output: ek $\in \mathbb{B}^{384k+32}$, dk $\in \mathbb{B}^{768k+96}$

$d, z \xleftarrow{\$} \mathbb{B}^{32}$
$(\rho, \delta) \leftarrow G(d \parallel k)$
$\hat{\mathbf{A}} \in (\mathbb{Z}_q^{256})^{k\times k} \leftarrow \mathrm{XOF}(\rho)$
$\mathbf{s} \in (\mathbb{Z}_q^{256})^{k} \leftarrow \mathrm{PRF}_{\eta_1}(\delta)$
$\mathbf{e} \in (\mathbb{Z}_q^{256})^{k} \leftarrow \mathrm{PRF}_{\eta_1}(\delta)$
$\hat{\mathbf{t}} \leftarrow \hat{\mathbf{A}} \circ \mathrm{NTT}(\mathbf{s}) + \mathrm{NTT}(\mathbf{e})$
ek $\leftarrow (\hat{\mathbf{t}} \parallel \rho)$
dk $\leftarrow (\hat{\mathbf{s}} \parallel \mathrm{ek} \parallel \mathrm{H}(\mathrm{ek}) \parallel z)$
return (ek,dk)

Algorithm 11 K-PKE.Encrypt($\mathrm{ek}_{\mathrm{PKE}}, m, r$) [4]

Input: encryption key $\mathrm{ek}_{\mathrm{PKE}} \in \mathbb{B}^{384k+32}$, message $m \in \mathbb{B}^{32}$, encryption randomness $r \in \mathbb{B}^{32}$
Output: ciphertext $c \in \mathbb{B}^{32(d_u k + d_v)}$

$\hat{\mathbf{A}} \in \mathbb{Z}_q^{k\times k} \leftarrow \mathrm{XOF}(\rho)$
$\mathbf{y} \in (\mathbb{Z}_q^{256})^{k} \leftarrow \mathrm{PRF}_{\eta_1}(r)$
$\mathbf{e_1} \in (\mathbb{Z}_q^{256})^{k} \leftarrow \mathrm{PRF}_{\eta_2}(r)$
$e_2 \in (\mathbb{Z}_q^{256}) \leftarrow \mathrm{PRF}_{\eta_2}(r)$
$\mathbf{u} \leftarrow \mathrm{NTT}^{-1}(\hat{\mathbf{A}}^{\top} \circ \mathrm{NTT}(\mathbf{y})) + \mathbf{e_1}$
$\mu \leftarrow \mathrm{Decompress}_1(\mathrm{ByteDecode}_1(m))$
$v \leftarrow \mathrm{NTT}^{-1}(\hat{\mathbf{t}}^{\top} \circ \hat{\mathbf{y}}) + e_2 + \mu$
$c_1 \leftarrow \mathrm{ByteEncode}_{d_u}(\mathrm{Compress}_{d_u}(\mathbf{u}))$
$c_2 \leftarrow \mathrm{ByteEncode}_{d_v}(\mathrm{Compress}_{d_v}(v))$
return $c \leftarrow (c_1 \parallel c_2)$

presumed intractability of the MLWE problem, a structured lattice problem believed to be quantum-resistant. ML-KEM follows a two-stage design paradigm. First, the hardness of the MLWE problem is leveraged to construct a public-key encryption (PKE) scheme, denoted K-PKE. Second, this PKE is transformed into a KEM using the Fujisaki–Okamoto (FO) transform [32], which strengthens chosen-ciphertext security. Importantly, K-PKE is not intended for standalone use; its algorithms operate only as internal subroutines within ML-KEM to ensure proper security guarantees.

The scheme comprises the standard three KEM algorithms: Keygen (Algorithm 10), Encaps (Algorithm 12), and Decaps (Algorithm 13). Each algorithm internally incorporates specific K-PKE operations as subroutines. It supports three parameter sets corresponding to the NIST security categories, determined by the variable k, which specifies the dimensions of the matrices and vectors used in the scheme. Key generation begins by sampling two uniformly random seeds (d, z). From d, two additional seeds are derived: the public seed ρ and secret seed σ. These are used in the internal K-PKE key generation process to produce: a decryption key, essentially a secret vector $\mathbf{s}$, and an encryption key—consisting of a public matrix $\mathbf{A}$ (generated deterministically from ρ) and a noisy vector $\mathbf{As} + \mathbf{e}$, where $\mathbf{e}$ is an error vector sampled from a small distribution. The encapsulation key is simply the encryption key from K-PKE, while the decapsulation key contains the decryption key, encryption key, a hash of the encryption key and an additional 32-byte random value for robustness.

The encapsulation procedure begins by validating the re-

Algorithm 12 ML-KEM.Encaps(ek) [4]

Input: encapsulation key ek $\in \mathbb{B}^{384k+32}$
Output: shared secret key $K \in \mathbb{B}^{32}$
ciphertext $c \in \mathbb{B}^{32(d_u k+d_v)}$

$m \xleftarrow{\$} \mathbb{B}^{32}$
$(K, r) \leftarrow G(m \parallel H(\text{ek}))$
$c \leftarrow$ K-PKE.Encrypt(ek,m,r)
return (K, c)

Algorithm 13 ML-KEM.Decaps(c,dk) [4]

Validated input: dk $\in \mathbb{B}^{768k+96}$, $c \in \mathbb{B}^{32(d_u k+d_v)}$
Output: shared key $K \in \mathbb{B}^{32}$

$(\hat{\mathbf{s}} \parallel \hat{\mathbf{t}} \parallel \rho \parallel h \parallel z) \leftarrow$ dk
$\mathbf{u}' \leftarrow \text{Decompress}_{d_u}(\text{ByteDecode}_{d_u}(c_1))$
$v' \leftarrow \text{Decompress}_{d_v}(\text{ByteDecode}_{d_v}(c_2))$
$w \leftarrow v' - \text{NTT}^{-1}(\hat{\mathbf{s}}^\top \circ \text{NTT}(\mathbf{u}'))$
$m' \leftarrow \text{ByteEncode}_1(\text{Compress}_1(w))$
$(K', r') \leftarrow G(m' \parallel h)$
$\bar{K} \leftarrow J(z \parallel c)$
$c' \leftarrow$ K-PKE.Encrypt(ek, m', r')
if $c \neq c'$ **then**
    $K' \leftarrow \bar{K}$
end if
return K'

ceived encapsulation key. A 32-byte message m is then generated using an approved random bit generator construction [24]. From m and the encapsulation key, both the shared secret key K and the encryption randomness are deterministically derived via a cryptographic hash function. The K-PKE encryption algorithm (Algorithm 11) is then used to encrypt m into the ciphertext c, embedding the necessary information to recover K upon decapsulation. The resulting ciphertext can be transmitted over an untrusted channel.

In reverse, the decapsulation procedure uses the decapsulation key and the ciphertext to recover the shared secret key. After the ciphertext and decapsulation key pass type and hash checks, these input components are used to obtain a plaintext m', as described step-by-step in Algorithm 13. The decapsulation algorithm then re-encrypts m' and computes a candidate shared secret key K' in the same manner as performed during encapsulation. Finally, decapsulation verifies whether the resulting ciphertext c matches the provided ciphertext c. In most cases, these ciphertexts will be identical, and the decapsulation process outputs the resulting shared secret key K'. Similar to other NIST PQC standardizations, ML-KEM employs four cryptographic functions from the SHA-3 family, described in detail in FIPS 202 [20], as internal hash functions (G, H and J), pseudorandom function (PRF) and extendable-output functions.

The correctness of ML-KEM lies in the ability of the decryption algorithm to recover the original message from the ciphertext, which serves as the preimage for the hash function used to derive the shared secret key. During encryption, the message m is 256-bit binary representation, with each bit upscaled by multiplying with $q/2$. The resulting ciphertext hides the message within noise terms and is expressed as: $\mathbf{u} = \mathbf{A}^\top\mathbf{y} + \mathbf{e}_1$ and $\mathbf{v} = \mathbf{t}^\top\mathbf{y} + e_2 + m$

During decryption, the receiver computes: $m' = \mathbf{v} - \mathbf{s}^\top\mathbf{u} = \mathbf{t}^\top\mathbf{y} + e_2 + m - \mathbf{s}^\top(\mathbf{A}^\top\mathbf{y} + \mathbf{e}_1) = \mathbf{e}^\top\mathbf{y} + e_2 + \mathbf{s}^\top\mathbf{e}_1 + m$

Since m was scaled during encryption, the additional noise terms remain small relative to the modulus. Thus, after downscaling to binary form, the recovered message satisfies $m' = m$. Finally, both encapsulation and decapsulation use the same recovered plaintext as input to the key-derivation hash function, ensuring that both parties agree on the shared secret key and completing the key establishment process. To further clarify the functioning of ML-KEM, a simple example with reduced parameters: $k = 2$, $n = 4$, and $q = 17$ is provided in [33]. We assume the following public matrix, secret vector, and error vector:

$$\mathrm{A} = \begin{pmatrix} 6x^3 + 16x^2 + 16x + 11 & 9x^3 + 4x^2 + 6x + 3 \\ 5x^3 + 3x^2 + 10x + 1 & 6x^3 + x^2 + 9x + a5 \end{pmatrix}$$

$\mathbf{e} = (\mathrm{x}^2, \mathrm{x}^2\text{-x})$ and $\mathbf{s} = (\text{-x}^3\text{-x}^2\text{+x}, \text{-x}^3\text{-x})$.

After the key generation procedure, the public vector $\mathbf{t}$ is derived as:

$$\mathbf{t} = (16x^3 + 15x^2 + 7, 10x^3 + 12^2 + 11x + 6)$$

Now, consider a plaintext message $m = 11(b'1011)$, which is represented as the polynomial $m = (x^3 + x + 1)$. Upscaling this message by multiplying with $\lfloor q/2 \rfloor = 9$ gives: $m_u = (9x^3 + 9x + 9)$. For the encryption step, assume the random polynomials are chosen as: $\mathbf{y} = (-x^3 + x^2, x^3 + x^2 - 1)$, $\mathbf{e}_1 = (-x^2 + x, x^2)$ and $e_2 = (-x^2 - x)$. The resulting ciphertext $(\mathbf{u}, \mathbf{v})$ is then computed as:

$$\mathbf{u} = (11x^3 + 11x^2 + 10x + 3, 4x^3 + 4x^2 + 13x + 11)$$
$$\text{and } \mathbf{v} = (7x^3 + 6x^2 + 8x + 15)$$

For decryption, the receiver computes:

$$m' = \mathbf{v} - \mathbf{s}^\top\mathbf{u} = (7x^3 + 14x^2 + 7 + 5)$$

After downscaling to the binary domain, we recover: $m' = (x^3 + x + 1)$, which corresponds to the original message m. Since both parties obtain the same recovered message, applying the key-derivation hash function on m ensures that they agree on the same shared secret key.

2) Hamming Quasi-Cyclic Overview and Explanation

HQC is a key encapsulation mechanism based on quasi-cyclic codes, designed such that no trapdoor is embedded within the code structure [34]. It leverages the structural advantages of quasi-cyclic codes while maintaining a direct security reduction to the problem of decoding a random linear code. Unlike other code-based candidates, HQC's security proof relies solely on parameterizations of the decisional quasi-cyclic syndrome decoding (QCSD) assumption. Structurally, HQC shares similarities with LWE-based cryptosystems, such as ML-KEM. As shown in Table 3 and Fig. 6, HQC exhibits significantly larger bandwidth requirements (encapsulation key and ciphertext) compared to ML-KEM, and its performance is relatively slower. Nevertheless, HQC was selected as a

TABLE 4. Parameter sets of HQC.

Instance	n_1	n_2	n	ℓ	ω	$\omega_r = \omega_e$	k	security	DFR	Reed-Solomon	Reed-Muller
hqc-128	46	384	17669	5	66	75	128	NIST-1	$< 2^{-128}$	[46, 16, 15]	[384, 8, 192]
hqc-192	56	640	35851	11	100	114	192	NIST-3	$< 2^{-192}$	[56, 24, 16]	[640, 8, 320]
hqc-256	90	640	57637	37	131	149	256	NIST-5	$< 2^{-256}$	[90, 32, 29]	[640, 8, 320]

Algorithm 14 HQC.KeyGen() [8]

Output: Encapsulation key ek_{KEM}, decapsulation key dk_{KEM}.

1: $\text{seed}_{\text{KEM}} \xleftarrow{\$} \mathcal{B}^{32}$
2: $(\text{seed}_{\text{PKE}}, \sigma) \leftarrow \text{XOF}(\text{seed}_{\text{KEM}})$
3: $(\text{seed}_{\text{PKE.dk}}, \text{seed}_{\text{PKE.ek}}) \leftarrow \text{I}(\text{seed}_{\text{PKE}})$
4: $(\mathbf{y}, \mathbf{x}) \leftarrow \text{SampleFixedWeightVect}_{\$}(\text{seed}_{\text{PKE.dk}}, \mathcal{R}_{\omega})$
5: $\mathbf{h} \leftarrow \text{SampleVect}(\text{seed}_{\text{PKE.ek}}, \mathcal{R})$
6: $\mathbf{s} \leftarrow \mathbf{x} + \mathbf{h}\cdot\mathbf{y}$
7: $\text{ek}_{\text{KEM}} \leftarrow (\text{seed}_{\text{PKE.ek}}, \mathbf{s})$
8: $\text{dk}_{\text{KEM}} \leftarrow (\text{ek}_{\text{KEM}}, \text{seed}_{\text{PKE.dk}}, \sigma, \text{seed}_{\text{KEM}})$
9: **return** $(\text{ek}_{\text{KEM}}, \text{dk}_{\text{KEM}})$

Algorithm 15 HQC.Encrypt(ek_{PKE}, $\mathbf{m}$, θ) [8]

Input: Encryption key ek_{PKE}, message $\mathbf{m}$, randomness θ.
Output: Ciphertext $\mathbf{c}_{\text{PKE}} = (\mathbf{u}, \mathbf{v})$.

1: $\mathbf{h} \leftarrow \text{SampleVect}(\text{seed}_{\text{PKE.ek}}, \mathcal{R})$
2: $\mathbf{r}_1, \mathbf{r}_2 \leftarrow \text{SampleFixedWeightVect}(\theta, \mathcal{R}_{\omega_r})$
3: $\mathbf{e} \leftarrow \text{SampleFixedWeightVect}(\theta, \mathcal{R}_{\omega_e})$
4: $\mathbf{u} \leftarrow \mathbf{r}_1 + \mathbf{h}\cdot\mathbf{r}_2$
5: $\mathbf{v} \leftarrow \mathcal{C}.\text{Encode}(\mathbf{m}) + \text{Truncate}(\mathbf{s}\cdot\mathbf{r}_2 + \mathbf{e}, \ell)$
6: **return** $\mathbf{c}_{\text{PKE}} = (\mathbf{u}, \mathbf{v})$

backup candidate for ML-KEM because it is based on computational hardness assumptions fundamentally different from those of ML-KEM, and it satisfies the NIST requirement of strong indistinguishability under adaptive chosen-ciphertext attack (IND-CCA2) security for a general-purpose KEM.

Similar to ML-KEM, HQC comprises three core algorithms: Key generation (Algorithm 14), Encaps (Algorithm 16) and Decaps (Algorithm 17), along with multiple parameter sets. The functional roles of these algorithms are essentially the same as in ML-KEM: enabling two parties to securely establish a shared secret key. The parameter sets for HQC, summarized in Table 4, consist of three variants that allow implementers to select an appropriate trade-off between security and performance. The HQC construction is based on an indistinguishability under chosen-plaintext attack (IND-CPA) secure PKE scheme, which in turn relies on the hardness of QCSD combined with the parity problem. To elevate this IND-CPA-secure PKE to an IND-CCA2-secure KEM, HQC employs the FO transformation [32], instantiated according to the HHK framework with implicit rejection, known as the HHK transform [35], which is specifically tailored for code-based cryptographic schemes.

Before delving deeper into its theoretical foundation, we first introduce the notations used throughout this section. Let $\mathbb{F}_2$ denote the binary finite field, $\mathcal{B}$ the set of unsigned 8-bit integers, and $\mathcal{R} = \mathbb{F}_2[X]/(X^n - 1)$ the quasi-cyclic ring on which the polynomials of HQC are defined. A linear code $\mathcal{C}$ of length n and dimension k is a k-dimensional subspace of $\mathcal{R}$; each element of $\mathcal{C}$ is referred to as a codeword. The notation $\omega(\cdot)$ represents the Hamming weight of a vector (i.e., the number of non-zero coefficients). Similar to ML-KEM, HQC relies on the SHA-3 family for its XOF and hash functions. In particular, the SHAKE256 instance is used for vector sampling, including both random vectors (SampleVect) and random fixed weight vectors (SampleFixedWeightVect). Additionally, four hash functions are employed with different domains: G and I are instantiated with SHA3-512, while H and J are instantiated with SHA3-256.

Key generation begins by generating the seed for both the KEM and its underlying PKE scheme. These seeds are used to uniformly sample a public vector $\mathbf{h}$ and two secret vectors $\mathbf{x}$ and $\mathbf{y}$ with fixed Hamming weights, as parameter is shown in Table 4. The syndrome $\mathbf{s}$ is then computed (line 5 of Algorithm 15) using arithmetic over the ring $\mathcal{R}$. The encapsulation key is composed of the encryption seed $\text{seed}_{\text{PKE.ek}}$ (which enables regeneration of $\mathbf{h}$) along with the syndrome, while the decapsulation key contains encapsulation key, $\text{seed}_{\text{PKE.dk}}$, which is used to regenerate secret vectors, random bitstring σ and the KEM seed seed_{KEM}.

The encapsulation process begins by receiving the encapsulation key and generating a uniformly distributed random message $\mathbf{m}$ and 128-bit public *salt*. The shared secret key and the randomness θ are then derived using the hash function G, which takes as input the concatenation of the hash image of ek_{KEM}, the message $\mathbf{m}$, and a salt. These values serve as inputs to the PKE encryption procedure, which produces the ciphertext in the HQC scheme. Here, the randomness θ is used as a seed to sample three error vectors $(\mathbf{e}, \mathbf{r}_1, \mathbf{r}_2)$ from the binary vector space, each with a fixed Hamming weight. These vectors mask the message and ensure IND-CPA security for the underlying PKE. The polynomial $\mathbf{u}$ is computed as described in line 4 of the algorithm, while the message $\mathbf{m}$ is encoded using a concatenation of shortened Reed–Solomon and duplicated Reed–Muller codes (RMRS). The construction of RMRS codes is parameterized according to the HQC specifications, summarized in Table 4. The public generator matrix $\mathbf{G}$ of the code $\mathcal{C}$ and the principles of RMRS encoding/decoding are described in [8]. The encoded message is then masked with structured noise, and the resulting bits are encoded to produce vector $\mathbf{v}$, as described in line 5 of Algorithm 15. Finally, after completing the PKE encryption routine, both the ciphertext of the KEM scheme and the

Algorithm 16 HQC.Encaps(ek_{KEM}) [8]

Input: Encapsulation key ek_{KEM}.
Output: Shared secret key K, ciphertext $\mathbf{c}_{\text{KEM}}$.

1: $\mathbf{m} \xleftarrow{\$} \mathcal{B}^{32}$. salt $\xleftarrow{\$} \mathcal{B}^{16}$
2: $(K, \theta) \leftarrow G(H(\text{ek}_{\text{KEM}})\|\mathbf{m}\|\text{salt})$
3: $\mathbf{c}_{\text{PKE}} \leftarrow \text{HQC.Encrypt}(\text{ek}_{\text{KEM}}, \mathbf{m}, \theta)$
4: $\mathbf{c}_{\text{KEM}} \leftarrow (\mathbf{c}_{\text{PKE}}, \text{salt})$
5: **return** $(K, \mathbf{c}_{\text{KEM}})$

Algorithm 17 HQC.Decaps(sk, **c**) [8]

Input: Decapsulation key dk_{KEM}, ciphertext $\mathbf{c}_{\text{KEM}}$.
Output: Shared secret key K'.

1: $\mathbf{y} \leftarrow \text{SampleFixedWeightVect}_{\$}(\text{seed}_{\text{PKE.dk}}, \mathcal{R}_\omega)$
2: $\mathbf{m'} \leftarrow \mathcal{C}.\text{Decode}(\mathbf{v} - \text{Truncate}(\mathbf{u}\cdot\mathbf{y}, \ell))$
3: $(K', \theta') \leftarrow G(H(\text{ek}_{\text{KEM}})\|\mathbf{m'}\|\text{salt})$
4: $\mathbf{c'}_{\text{PKE}} \leftarrow \text{HQC.Encrypt}(\text{ek}_{\text{KEM}}, \mathbf{m'}, \theta)$
5: $\mathbf{c'}_{\text{KEM}} \leftarrow (\mathbf{c'}_{\text{PKE}}, \text{salt})$
6: **if** $\mathbf{m'} = \perp$ or $\mathbf{c'}_{\text{KEM}} \neq \mathbf{c}_{\text{KEM}}$ **then**
7: $K' \leftarrow J(H(\text{ek}_{\text{KEM}})\| \sigma \|\mathbf{c}_{\text{KEM}})$
8: **end**
9: **return** K'

corresponding shared secret key are obtained.

In reverse, the decapsulation process recovers the message **m'** through RMRS decoding applied to (**v** - **u**·**y**). The recovered message is then processed in the same manner as in the encapsulation phase to regenerate the ciphertext $\mathbf{c'}_{\text{KEM}}$. If the regenerated ciphertext $\mathbf{c}'_{\text{KEM}}$ matches the received ciphertext $\mathbf{c}_{\text{KEM}}$, the receiver successfully reconstructs the correct shared secret key. Otherwise, the algorithm performs an "implicit rejection": the candidate key K' is replaced with a pseudo-random value derived via a hash function, as specified in line 7 of Algorithm 17.

As a code-based KEM, the correctness of the HQC scheme fundamentally relies on the decoding capability of the code $\mathcal{C}$ and its associated $\mathcal{C}$.Encode and $\mathcal{C}$.Decode algorithms, which can reliably correct up to $\triangle$ errors. Specifically, $\mathcal{C}$ successfully decodes $\mathbf{v}$-$\mathbf{u}$·$\mathbf{y}$ whenever the following conditions hold:

$$\begin{cases} \omega(\mathbf{s}\mathbf{r}_2 - \mathbf{u}\mathbf{y} + \mathbf{e}) \leq \triangle \\ \omega((\mathbf{x} + \mathbf{h}\mathbf{y}) \cdot \mathbf{r}_2 - (\mathbf{r}_1 + \mathbf{h}\mathbf{r}_2)\mathbf{y} + \mathbf{e}) \leq \triangle \\ \omega(\mathbf{x}\mathbf{r}_2 - \mathbf{r}_1\mathbf{y} + \mathbf{e}) \leq \triangle \end{cases} \tag{4}$$

The HQC specification [8] provides a precise analysis of the error distribution associated with the effective error vector **e'** = **x**·$\mathbf{r}_2$ - $\mathbf{r}_1$·**y** + **e**. Based on this analysis, the decoding failure rate (DFR) of the scheme is both theoretically estimated and empirically simulated, ensuring that the resulting DFR aligns with the required security levels specified in Table 4.

III. PQC IMPLEMENTATIONS ACROSS PLATFORMS

A. PQC GPU-ACCELERATED IMPLEMENTATIONS

Graphics processing units (GPUs) are specialized processors originally designed to accelerate complex mathematical and graphical computations. Unlike central processing units (CPUs), which are optimized for sequential processing and general-purpose tasks, GPUs feature a massively parallel architecture with thousands of lightweight cores capable of executing many operations simultaneously. This architectural difference makes GPUs particularly suitable for high-throughput implementations of computationally intensive algorithms. Leveraging this parallelism, many GPU-accelerated implementations of PQC algorithms have been proposed, showing substantial reductions in execution time and significant improvements in throughput compared with CPU-based implementations. To the best of our knowledge, we summarize the most notable GPU-based PQC works in [36]–[46], and provide detailed implementation results in Table 5. Unlike CPUs, which are constrained by the Von Neumann architecture and memory bottlenecks, modern GPUs are built from clusters of streaming multiprocessors (SMs), each containing multiple cores, arithmetic logic units (ALUs), and local instruction caches, as illustrated in Fig. 7. This architecture enables fine-grained parallelism and efficient execution of operations that can be distributed across thousands of threads, making GPUs highly effective for accelerating PQC algorithms.

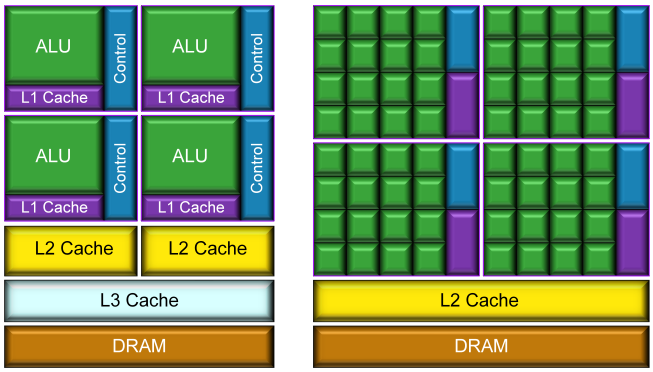

FIGURE 7. Comparison of CPU and GPU architecture.

From the surveyed PQC algorithms, it is evident that polynomial arithmetic and hash-based operations dominate the computational workload. As a result, GPU acceleration typically focuses on optimizing these operations to maximize performance. In lattice-based cryptography, such as ML-DSA and ML-KEM, polynomial arithmetic is often performed using the NTT, which maps efficiently onto GPU architectures. Similarly, FALCON benefits from GPU acceleration through parallelized NTT and FFT operations. Notably, Shen et al. [36] proposed GPU implementations of Dilithium on high-end NVIDIA devices such as the Tesla A100 and RTX 4090, reporting speedups of more than 166× compared to single-threaded CPU implementations. The implementation is publicly available at https://github.com/encryptorion-lab/cuDilithium, providing a valuable resource for further research and analysis. Their strategy exploits GPU parallelism by distributing arithmetic operations (e.g., NTT, polynomial multiplication) and hash-based sampling across multiple threads. A similar strategy was adopted by Ji et al. [42], who developed a high-performance GPU-accelerated Kyber im-

TABLE 5. Throughput (OP/s) of PQC implementations on GPU in the highest security level.

Ref	Scheme	Device	Keygen	Encryp/ Sign	Decryp/ Verify
[36]	Dilithium	RTX 4090	888,971	488,006	964,374
[37]	ML-DSA	RTX 4090	-	785,277	-
[42]	Kyber	RTX 3080	160,130	187,832,	539,450
[38]	Falcon	RTX 3090	1,972	167,928	997,067
[39]	Falcon	RTX 3080	-	15,239	1,217,317
[41]	Falcon	RTX 4090	-	420,250	10,311,039
[44]	SPHINCS$^+$	RTX 3090	100,160	11,401	106,280
[45]	SPHINCS$^+$	RTX 3090	-	21,395	779,879

plementation on devices such as the Tesla V100 and GeForce RTX 3080. Their approach also emphasized maximizing parallel execution for polynomial arithmetic, achieving notable performance gains over CPU implementations.

Similar to ML-DSA and ML-KEM, the FALCON algorithm also benefits significantly from GPU acceleration, as its core operations—FFT, NTT, and polynomial arithmetic—map efficiently onto massively parallel GPU architectures. Several studies [38]–[41] have explored this approach. For example, the work in [38] employed CUDA with 32 concurrent streams on an NVIDIA GeForce RTX 3090, demonstrating more than a 27× throughput improvement compared to a high-end CPU implementation on the AMD Ryzen 9 5900X (4.7 GHz) using AVX2 instructions. A subsequent study [41] further improved performance by fully exploiting the GPU's parallel core architecture, carefully optimizing memory access and kernel execution to reach near-peak hardware utilization. The detailed implementation results are summarized in table 5, clearly illustrating the scalability of FALCON when mapped onto modern GPU platforms.

For hash-based signature schemes, the inherent tree-based structure also allows for parallelization across GPU cores. In particular, functions such as FORS, XMSS, and WOTS+, which dominate the computational workload in SPHINCS+, are well-suited for GPU acceleration. The study in [44] proposed a GPU-accelerated SPHINCS+ implementation on an NVIDIA RTX 3090, utilizing 512 CUDA blocks, with each block launching 512 threads for FORS, 16×16 threads for Hypertree Merkle signature computations, and 8×67 threads for WOTS+ functions. This parallelization strategy achieved a speedup of approximately 25× compared to 8-thread AVX2 CPU implementation on AMD Ryzen 9 5900X (3.7 GHz). More recently, Wu et al. [45] further optimized SPHINCS+ by leveraging warp-level primitives, shared memory optimization, and improved workload scheduling to maximize GPU resource utilization. Their implementation achieved an additional 2× performance improvement over the previous state-of-the-art in [44], underscoring the effectiveness of GPU acceleration in scaling hash-based PQC schemes.

TABLE 6. Performance results (cycle count) of PQC implementations on resource-constrained devices in the highest security level.

Ref	Scheme	Device	Keygen	Encaps/ Sign	Decaps/ Verify
[50]	Dilithium	Raspberry Pi 4	782,752	1,436,988	764,886
	Kyber		156,694	192,280	184,161
[51]	Dilithium	STM32F407G	4,829k	9,037k	4,718k
	Kyber		1,138k	1,324k	1,227k
[52]	Dilithium	ODROID-XU4	1,256k	2,652k	1,260k
	Kyber		294,000	348,000	339,000
[56]	FALCON	Jetson Xavier	29,956k	990,487	62,547
[58]	SPHINCS$^+$-f	Nucleo-L4R5ZI	59,808k	1,281,329k	32,947k
	SPHINCS$^+$-s		952,800k	12,304,133k	16,715k
[59]	HQC	Nucleo-L4R5ZI	29,830k	59,650k	93,150k

B. PQC IMPLEMENTATIONS ON RESOURCE-CONSTRAINED DEVICES

Public-key cryptography underpins a wide range of real-world applications, from secure communications to authentication and key management. While GPU-accelerated implementations of PQC are crucial for achieving high throughput in data centers and cloud environments, it is equally important to ensure efficient implementations on resource-constrained embedded devices. In edge-network scenarios—such as IoT devices, low-power sensors, and mobile systems—the direct execution of computationally intensive PQC algorithms can impose significant performance and energy overheads. These limitations may hinder the practical adoption of PQC in such environments, highlighting the need for careful optimization not only at the algorithmic level but also in terms of lightweight protocol design, memory usage, and hardware-software co-design. Ensuring that PQC can run efficiently on embedded platforms is therefore essential to its deployment in real-world applications, where billions of devices must operate securely under strict power and resource constraints. During the PQC standardization process, Kannwischer et al. [47] made a major contribution by integrating suitable PQC implementations into the pqm4 benchmarking framework, targeting 32-bit ARM Cortex-M microcontrollers. They provided comprehensive benchmarking results on the STM32L4R5ZI platform, which features a Cortex-M4 core with 640 KB of RAM, a widely used class of embedded device. Their project currently includes implementations of 15 promising PQC submissions, with publicly available source code [47].

Beyond pqm4, several optimized PQC implementations on resource-constrained platforms have been proposed [48]–[59], summarized in Table 6. Notably, Seo et al. [48] presented an optimized software implementation of Dilithium on NVIDIA's Jetson AGX Xavier, a heterogeneous embedded system-on-chip widely used in robotics and edge AI applications. Their results demonstrate that PQC can be efficiently integrated into embedded AI platforms, where both performance and energy efficiency are critical. Similarly, the work in [50] provides publicly available source code and bench-

marking results for Dilithium and Kyber on the Arm Cortex-A72 processor as well as Apple's high-end M1 desktop core, highlighting the portability of optimized PQC across different Arm-based and heterogeneous platforms. By leveraging parallelization strategies and NEON SIMD engine available in Arm Cortex-A processors, the study in [52] achieved more than 3× performance improvement for Dilithium and Kyber compared to pqm4-based designs [49], [51]. This demonstrates the importance of architecture-specific optimizations in bridging the performance gap for constrained devices.

For the backup NIST PQC candidates, there are also several notable resource-constrained implementations. FALCON has been efficiently deployed on the Jetson Xavier platform [56], showcasing the benefits of GPU-equipped embedded SoCs for lattice-based signatures. Hash-based schemes such as SPHINCS+ have been implemented on the Arm Cortex-M4 processor using the NUCLEO-L4R5ZI development board [58], illustrating that even highly demanding hash-based PQC can be adapted to lightweight devices. Additionally, the HQC implementation on the NUCLEO-L4R5ZI board [59] further emphasizes the feasibility of running code-based PQC schemes on low-power microcontrollers.

C. PQC IMPLEMENTATIONS ON FPGA

Field-programmable gate arrays (FPGAs) are reconfigurable integrated circuits that enable the rapid development of custom logic for both prototyping and deployment of final system designs. Thanks to their programmable fabric, FPGAs can be quickly programmed—and reprogrammed—to perform application-specific functions without the need for costly and time-consuming ASIC fabrication. As illustrated in Fig. 8, FPGAs represent a promising hardware platform for implementing PQC algorithms, offering advantages in performance, energy efficiency, cost-effectiveness, and resilience against side-channel attacks (SCA) compared to software-only implementations. Unlike software executed on general-purpose CPUs, where instructions are processed sequentially, FPGA architectures allow true parallel execution of operations at the hardware level, drastically reducing execution latency. Moreover, while software optimizations are constrained by fixed processor instruction sets, FPGAs provide the ability to design application-specific data paths and arithmetic units, delivering performance and efficiency tailored precisely to the structure of PQC algorithms.

The overall high-level PQC hardware architecture is illustrated in Fig. 9. It typically consists of a controller, arithmetic module, hashing and sampling module, internal memory, an interconnect network, an AXI stream interface for communication with the external environment, and several scheme-specific functional components. Depending on the requirements of each PQC algorithm, the detailed configuration of these submodules may vary. Furthermore, the inherent flexibility of FPGA architectures allows designers to tailor PQC implementations to diverse use cases and optimization targets. On the one hand, lightweight architectures can be deployed on resource-constrained or low-power devices, while

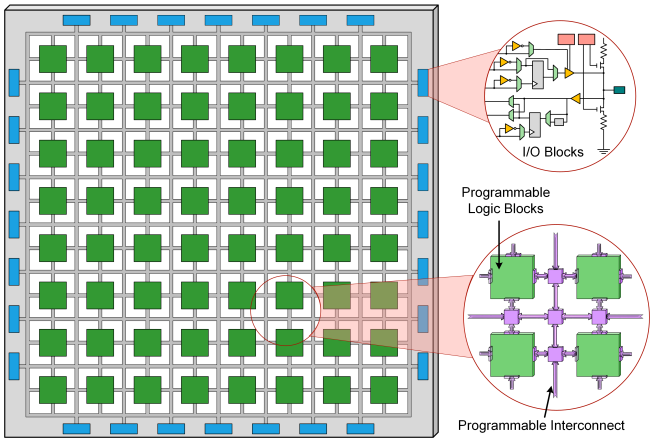

FIGURE 8. FPGA architecture.

on the other hand, high-performance designs can maximize throughput, reduce execution latency and even support the parallel processing of multiple cryptographic tasks. This flexibility also extends to architectural strategies: designers may instantiate multiple dedicated submodules for specific operations to boost performance, or unify several processes into a single versatile submodule to minimize resource usage. The decision between these approaches is ultimately guided by the implementation goals, whether emphasizing high throughput or area and cost efficiency.

Moreover, FPGA-based implementations, with their reconfigurable and flexible architectures, provide stronger resistance against a wide range of SCAs compared to purely software-based solutions. For instance, power analysis attacks exploit variations in power consumption during cryptographic operations—a particular vulnerability in single-core CPU designs. High-performance FPGA architectures can mitigate this threat by leveraging parallelism and task-level balancing, thereby obscuring power consumption patterns [121]. Similarly, electromagnetic analysis, which monitors device radiation to extract sensitive information, can be countered through custom hardware layouts and parallel execution strategies that reduce signal leakage. By utilizing on-chip memory, FPGA designs also eliminate vulnerabilities to cache-based attacks, which exploit CPU cache access patterns, hits, and misses [122]. Furthermore, timing attacks can be mitigated in FPGA implementations by enforcing constant-time execution for critical operations or by masking sensitive information through simultaneous parallel processes. Together, these capabilities make FPGA-based cryptographic accelerators a robust platform for PQC security.

Recognizing the potential of FPGA accelerators for PQC, recent research has introduced a wide range of efficient hardware architectures and FPGA-based implementations. Among these, the primary PQC algorithms standardized for KEM and DSA (ML-KEM and ML-DSA) have received more extensive and mature FPGA design efforts compared to the backup PQC candidates. To ensure clarity and concise-

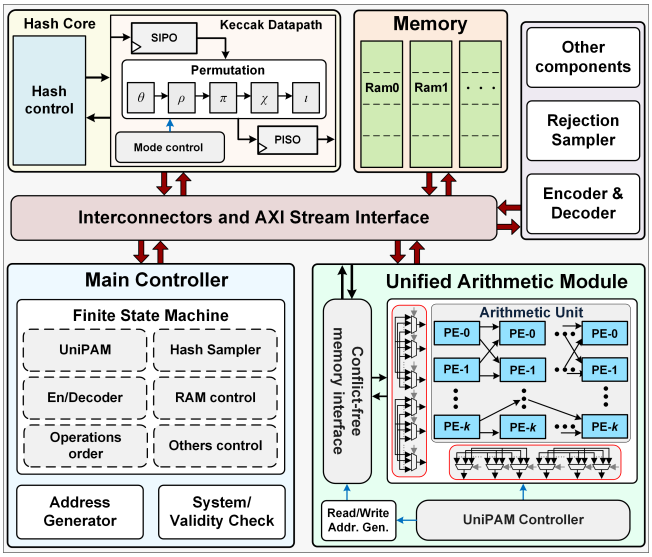

FIGURE 9. Overall high-level PQC hardware architecture.

TABLE 7. Summary of the NTT architectures for ML-KEM and ML-DSA.

Ref	Structure	Resource Utilization				Freq[1]	Latency[2]
		LUT	FF	DSP	RAM	(MHz)	(CCs)
[61]^k	R2MDF	3,918	4,292	26	1	265	64/64
[62]^k	R2MDC	4,834	4,683	0	1	250	128/128
[62]^d	R2MDC	7,451	5,275	0	0	180	128/128
[62]^d	R4MDC	13,804	11,019	0	0	163	64/64
[67]^c	F-R2MDC	4,374	1,900	8	1	325	256^d/128^k
[63]^k	Radix-2	2,543	792	4	9	182	232/233
[64]^k	Radix-2	2,081	2237	0	0	304.7	504/504
[94]^k	Rad.-2/4	1,740	643	4	0	200	224/224
[65]^d	Radix-4	2,637	1,071	8	1	385	268/268
[66]^k	Rad.-8/16	9,985	4,580	29	12	132	58/58
[66]^d	Radix-16	12,720	7,834	50	14.5	183	58/58
[68]^c	Radix-2	3,105	2,506	8	0	357	264^d/120^k
[71]^f	Radix-2	17,395	7,950	20	4	134	2048^I/4096^V

^{k}/^{d}/^{c}/^{f}: Supporting for NTT of Kyber/ Dilithium/ combined ML-KEM and ML-DSA/ FFT for secret level I and V of FALCON.
[1]: Except [65], [68] installed on Zynq US+ and [67] on Virtex US+, others on Artix-7 platform.
[2]: Computation time of NTT/INTT or FFT/IFFT for one polynomial.

ness, we summarize FPGA implementations of the primary PQC standards in Table 8, while FPGA implementations of the backup candidates are listed in Table 9. The hardware implementations for ML-KEM and ML-DSA included in this survey are carefully selected to highlight the most significant and representative contributions. Specifically, we focus on proposals targeting the Round 3 NIST candidates Kyber and Dilithium, as well as the finalized ML-KEM and ML-DSA standards. Earlier studies that predate Round 3 or fail to provide configurable support for all main algorithms of Kyber and Dilithium are excluded to maintain conciseness. In contrast, for hardware implementations of the backup NIST standards, the number of available works is relatively small, and many designs are incomplete. For example, no proposal has yet provided a fully configurable implementation covering all processes of FALCON. Therefore, to provide a comprehensive view of the state of this research direction, we include all notable and relevant studies available to date.

1) Optimizing important submodules

Within the selected NIST PQC standards, the underlying security assumptions rely on lattice-based, code-based, and hash-based cryptography. Consequently, arithmetic computations and hashing processes dominate the execution time of PQC algorithms. Based on the best of our knowledge, the most significant submodule optimizations proposed in the literature include NTT architectures for ML-KEM and ML-DSA [61]–[68], FFT and unified FFT/NTT architectures for FALCON [70]–[72], and efficient sparse polynomial multiplication accelerators for HQC [73]. For hash functions, which are indispensable across all PQC schemes for hashing and pseudorandom sampling, standardized cryptographic primitives are described in FIPS 202 [20]. Since the design of hash modules is both necessary and challenging, FPGA implementations frequently present their own optimized solutions. However, a closer survey of the literature reveals that the majority of studies adopt, modify, or reference two open-source RTL designs of SHA-3. The first is provided by the Keccak team [74], which introduces VHDL implementations of high-speed, mid-range, and low-area cores tailored to specific hardware targets. The second is from the CERG team [75], offering a configurable implementation capable of supporting multiple SHA-3 variants within a single core. In addition, SLH-DSA incorporates a variant that employs SHA-2 [23] for instantiating its hash functions, and both hardware and software accelerators for SHA-2 have been extensively studied in [76].

In lattice-based schemes such as ML-DSA, ML-KEM, and FALCON, the NTT and FFT serve as efficient optimizations to reduce the complexity of polynomial multiplication, which is one of the most computationally intensive operations in these algorithms. Leveraging the flexibility of FPGA architectures, NTT and FFT modules can be tailored either to maximize performance or to minimize hardware resource utilization, depending on design goals. Generally, NTT and FFT architectures can be classified into two main categories: (i) fully-pipelined architectures and (ii) in-place iterative architectures [60]. Approach (i), fully-pipelined architectures, process all stages sequentially and produce outputs every clock cycle, thereby achieving high throughput. Fig. 10 illustrates the overall structure of a radix-2 multipath delay commutator (MDC) architecture, which represents typical fully-pipelined design. This approach has been applied, for example, to the NTT implementations of Kyber and Dilithium in [62]. Another well-known fully-pipelined design is the multipath delay feedback (MDF) architecture, introduced for Kyber in [61]. Both MDC and MDF exemplify high-performance implementations that prioritize execution speed and throughput,

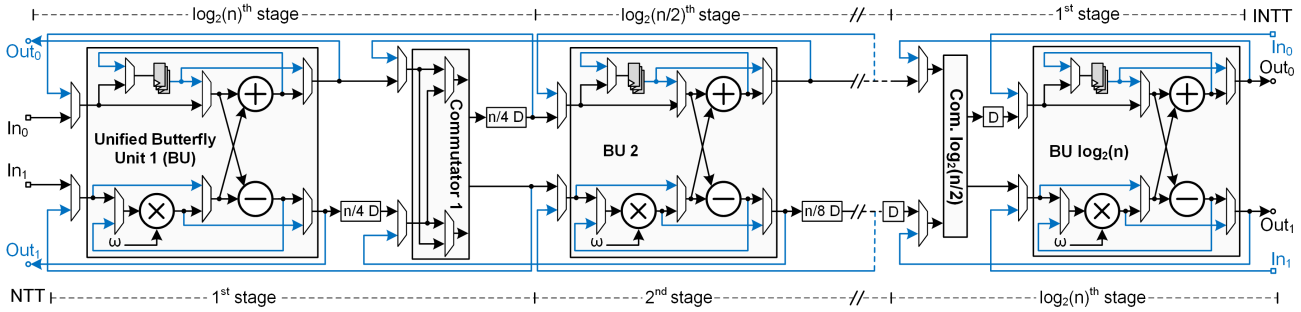

FIGURE 10. Overall fully-pipelined NTT/FTT architecture.

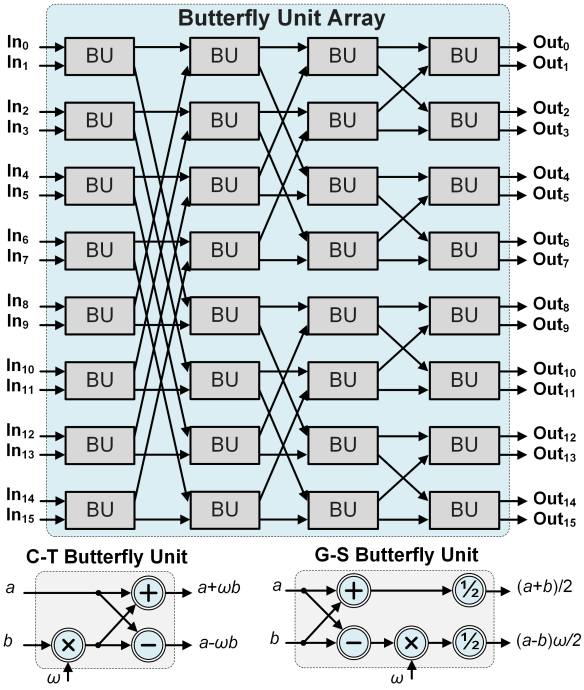

FIGURE 11. Overall in-place NTT/FTT architecture [60].

making them well-suited for time-critical applications.

In contrast, approach (ii), in-place iterative architectures, rely on one or multiple processing elements (PEs) to perform computations iteratively, stage by stage. Polynomial samples are stored in memory during execution, enabling compact, resource-efficient designs and making this approach ideal for integration with polynomial arithmetic operations. A typical example is the radix-2 butterfly unit (BU) architecture proposed in [63], which applies the classical Decimation-In-Time Cooley–Tukey (CT) algorithm for forward NTT/FFT and the decimation-in-frequency Gentleman–Sande (GS) algorithm for the inverse transform. To further enhance performance, higher-radix methods have been explored. For instance, a radix-4 design was proposed in [65], while radix-8 and radix-16 implementations were introduced in [66]. Fig. 11 shows an example of a radix-2^4 iterative pipelined architecture [60], which computes four NTT stages within a single pipeline and offers significant performance improvements over the conventional radix-2 method. In addition, folding-pipelined architectures [67], [69] provide an effective compromise between high throughput and low hardware resource consumption, making them attractive for practical FPGA-based PQC implementations.

Recognizing the importance of arithmetic units (AUs) in PQC schemes, we provide a comprehensive summary of the most notable and efficient hardware architectures implementing the NTT for ML-DSA and ML-KEM, as well as FFT in FALCON, in Table 7. Given the same lattice-based structure of these schemes, combined architectures that unify the AUs for both primary PQC algorithms have also become an important research direction, with notable examples including the folding pipelined architecture in [67] and the multiple iterative radix-2 architecture in [68]. At a finer level, the processing element architecture must also be carefully considered to achieve fully optimized designs. Detailed architectural techniques and optimization strategies for BU are well described across existing studies on AUs [61]–[68]. Recently, K. Javeed et al. [77] proposed a configurable unified BU supporting lattice-based cryptosystems. Enhancing and optimizing these submodules forms the foundational basis for improving overall hardware efficiency in PQC schemes—reducing execution time, lowering resource utilization and mitigating latency bottlenecks in complex operations.

2) Notable FPGA implementations for primary PQC schemes

As the primary PQC schemes were announced, numerous studies have proposed both area-efficient and high-performance hardware architectures for ML-DSA and ML-KEM. Over time, these proposals have become increasingly refined and complete, with later works introducing significant innovations to improve implementation results. The designs are typically evaluated using standard metrics such as execution time, hardware utilization, and the area-time product (ATP). To illustrate this evolution, we summarize the implementation analyses, claimed novelties, and notable characteristics of the most representative studies in Table 8. In particular, the high-performance architecture for ML-DSA in [87], as well as the area-efficient ML-KEM implementations in [100] and [101], stand out as the most relevant to the current

TABLE 8. Summary of hardware implementations for primary PQC schemes.

Ref	Scheme	Device[1]	Resource[2] LUT/FF/DSP/BRAM	Freq. (MHz)	Time[3] (kCCs)	Novelty claims and notable characteristics
[78]	Dilithium	VUS+	K: 54k/25k/182/15 S: 68.5k/86.3k/965/145 Ve: 61.7k/35.0k/316/18	350 333 158	12.6/18.3/10.5 18.2/21.0/15.0 23.0/22.4/20.2	First FPGA architectures for Round 3 Dilithium, with three independent designs targeting the main algorithms: Keygen, Sign, Verify.
[79]	Dilithium	Artix-7	II: 27.4k/10.7k/45/15 III: 30.9k/11.4k/45/21 V: 44.7k/13.8k/45/31	163 145 140	18.8/76.6/19.7 33.1/123.2/32.1 51.0/146/51.7	Propose the first configurable Dilithium architectures for each of the three security levels; propose a unified arithmetic architecture integrating multiple type of polynomial arithmetic operations in Dilithium.
[80]	Dilithium	VUS+	53.9k/28.4k/16/29	256	4.9/29.9/6.6 8.3/49.4/9.7 14/55.1/14.6	Proposed the first configurable Dilithium architecture supporting all three security levels and main phases. Proposed using multiple unified arithmetic and hash modules to improve performance.
[81]	Dilithium	Artix-7	30.0k/10.4k/10/11	97	4.2/31.6/4.4 5.9/49.5/6.2 8.8/55.3/9.04	Proposed a compact Dilithium architecture supporting all phases and security levels. Proposed a segmented pipelined processing method to reduce both storage requirements and processing time.
[82]	Dilithium	Z-7000	II: 18.6k/7.3k/10/17 III: 19.6k/8.5k/10/21 V: 21.0k/9.7k/10/28	159	7.8/52.0/7.7 13.0/89.2/11.2 20.2/93.7/15.9	Proposed HW/SW Dilithium implementations for each security level. Propose the arithmetic unit based on radix-4 NTT method. Corrected recursive reduction method in modular multiplication proposed in [79].
[83]	Dilithium	VUS+	V: 14.0k/6.85k/4/35	391	63.2/113.9/67.9	Propose a lightweight architecture for FPGA and ASIC (TSMC 65nm).
[85]	Dilithium	VUS+	32.3k/9.7k/24/14	300	4.4/30.1/5.0 7.3/49.5/8.0 10.9/55.2/12.7	Propose a high-speed Dilithium hardware accelerator with low-latency NTT butterfly arithmetic unit based on the Karatsuba algorithm.
[87]	ML-DSA	VUS+	49.8k/23.5k/16/27.5	300	3.3/29.2/4.1 5.3/46.3/5.85 8.2/50.85/8.1	Proposed the first configurable ML-DSA architecture supporting for all three security levels and main phases. Proposed a flexible unified arithmetic module and dedicated hash modules for each SHAKE variant.
[88]	Dilithium	Artix-7	52.9k/31.2k/38/11	191	4.8/28.8/5.2 7.9/48.6/8.5 11.8/54.7/12.5	Improved parallel modular multiplication and high-speed polynomial multiplier. Implemented a fully pipelined MDC NTT architecture for unified arithmetic module.
[89]	Kyber	Artix-7	Ser: 7.4k/4.64k/2/3 Cl: 6.8k/3.98k/2/3	161 167	3.8/5.1/6.7 6.3/7.9/10.0 9.4/11.3/13.9	First client/server implementation for Round 3 Kyber: server handles KeyGen and Decaps processes, client handles Encaps. Proposed compact unified arithmetic module handle all Kyber arithmetic operations.
[90]	Kyber	Artix-7	18.0k/5.0k/6/15 16.0k/6.0k/9/16 16.0k/6.0k/12/17	115 115 112	4.0/7.0/10.0 7.0/10.0/14.0 10.0/14.0/18.0	Proposed the first configurable implementations of three main Kyber algorithms for each security level in FPGA and ASIC (TSMC 65nm).
[92]	Kyber	Artix-7	9.3k/8.2k/4/6 10.4k/9.5k/8/6 11.5k/10.6k/8/10.5	220	2.1/3.3/4.5 2.7/3.9/5.0 3.6/4.8/6.0	Proposed high-speed hardware architectures for Kyber using multiple parallel butterfly units in the arithmetic module. Performance result is significantly higher than previous proposals in [89], [90].
[93]	Kyber	ZUS+	Ser: 17.8k/14.0k/2/0 Cl: 15.5k/12.5k/2/0 (in security level 5)	435 435	2.7/3.4/4.1 (Sec_level 5)	Proposed a high-performance hardware accelerator for Kyber, with three server/client designs for each security level. Proposed using MDC NTT architecture and multiplier without DSP for the arithmetic unit.
[94]	Kyber	Artix-7	7.3k/3.2k/4/5	200	2.4/3.0/4.2 4.2/5.0/7.0 6.8/8.0/10.2	Proposed the first configurable Kyber architecture supporting all three security levels and main phases. Proposed a mixed radix-4/2 method for implementing NTT/INTT in the arithmetic module.
[95]	Kyber	Artix-7	6.9k/3.3k/2/3	200	2.9/3.9/5.2 5.0/6.3/7.8 8.0/9.0/11.2	Compact architecture supporting all Kyber phases and security levels. Proposed a split-radix method to integrate all arithmetic operations in a unified arithmetic module with low area utilization.
[98]	Kyber	Artix-7	8.1k/6.1k/4/5.5	311	2.5/3.7/4.6 4.1/5.6/6.9 6.3/8.2/9.9	Propose a high-performance hardware controller for Kyber, achieving a significant breakthrough in operating frequency and delivering improved overall performance with lower resource consumption.
[99]	ML-KEM	Artix-7	25.8k/18.6k/21/14	210	1.8/1.9/3.2 2.6/2.7/4.1 3.4/3.5/5.8	Propose the first configurable architecture supporting all main algorithms and security levels of ML-KEM.

TABLE 8. (Continued) Summary of hardware implementations for primary PQC schemes.

Ref	Scheme	Device[1]	Resource[2] LUT/FF/DSP/BRAM	Freq. (MHz)	Time[3] (kCCs)	Novelty claims and notable characteristics
[100]	ML-KEM	Artix-7	7.6k/5.7k/4/3	169	1.8/2.3/3.6 3.1/3.6/5.6 4.5/5.1/7.6	Propose a low-area, efficient configurable architecture for ML-KEM on FPGA (Artix-7 family) and ASIC (180nm CMOS) fabrication.
[101]	ML-KEM	Artix-7	12.0k/6.9k/4/9	220	1.8/2.7/4.0 3.1/4.1/5.8 4.6/6.1/8.0	Propose a configurable architecture for ML-KEM on FPGA and ASIC (14nm technology). Proposed efficient timing schedules for the three main processes of ML-KEM.

[1] VUS+: Virtex UltraScale+; Z-7000: Zynq-7000; ZUS+: Zynq UltraScale+

[2] K: Keygen; S: Signature generation; Ve: Verification; II, III, V: security levels. Ser: server side; Cl: client side.

[3] The 3 lines corresponding with time execution of 3 security levels rising from low to high, in order of: Keygen, Sign/Encaps, Verify/Decaps.

NIST standards. These designs not only support all phases of DSA and KEM schemes but also allow dynamic selection of the security level at runtime. Such proposals are promising candidates for practical deployment in the near future.

Furthermore, the concept of a unified architecture for ML-KEM and ML-DSA is highly effective, given the similarity of their underlying lattice-based structures, as demonstrated in recent crypto-agility projects for both primary NIST PQC standards [102]–[105]. In particular, the lightweight unified architectures presented in [103] and [104] demonstrate that both schemes can be efficiently implemented on FPGAs with low-area requirements while still maintaining reasonable execution times. Notably, the study in [105] proposes a high-performance unified architecture based on the final ML-DSA and ML-KEM parameter sets and validates the design using the official test vectors provided by the NIST Cryptographic Algorithm Validation Program [106]. These designs leverage the shared MLWE-based construction, where both schemes rely heavily on arithmetic computations and hash-function operations. From an industrial perspective, iWave Global has partnered with Xiphera to deliver quantum-resistant security on the Agilex™ 5 System-on-Module (SoM) powered by Altera [107]. By integrating Xiphera's xQlave IP cores for ML-KEM and ML-DSA, the Agilex™ 5 SoM enables real-time, quantum-secure key exchange and digital signature operations directly in FPGA hardware. Such versatile approaches highlight the practicality of implementing multiple PQC standards within a single hardware design, significantly reducing both cost and engineering effort when deploying PQC accelerators in real-world applications.

3) FPGA implementations for backup PQC schemes

Compared with the primary PQC standards, the backup schemes are generally regarded as more complex and challenging to implement in hardware. Their complexity arises from their underlying security assumptions: SLH-DSA is hash-based, FALCON is a lattice-based signature scheme over NTRU, and HQC is a code-based KEM. This increased difficulty can also be observed in the CPU benchmarks presented in Fig. 3 and Fig. 6, where the execution times of backup finalists are significantly longer than those of the corresponding primary schemes for both DSA and KEM categories. Consequently, hardware implementation proposals for these algorithms remain limited and are not yet fully comprehensive in supporting all processes and variants within a single design. To date, our survey of prestigious journals and conference papers has identified FPGA implementations of SLH-DSA in [108]–[112], FALCON in [113]–[115], and HQC in [116]–[120]. The most representative proposals and their notable characteristics are summarized in Table 9.

For SLH-DSA, the work in [108] presented the first hardware implementation of SPHINCS+ for SHAKE256-based variants. This design supports both signing and verification and provides six independent implementations across different security levels, targeting simple and robust SPHINCS+ variants. Later, in 2021, Berthet et al. [109] introduced an area-efficient design for SHA256-based SPHINCS+ at security levels I and V. Their implementation supports all three main processes within a lightweight architecture, significantly reducing area requirements compared to prior work. Independent implementations for each SLH-DSA variant on the Artix-7 FPGA family were reported in [110] and [111]. More recently, Huang et al. [112] proposed a reconfigurable processor supporting three SHAKE256-f SPHINCS+ variants across different security levels.

Due to the inherent complexity of implementing FALCON in hardware, only a limited number of incomplete designs have been proposed. Beckwith et al. [113] focused on the Verify operation, which is the simplest and fastest process in the FALCON construction. The work in [114] employed high-level synthesis (HLS) to implement the main algorithms of FALCON. While the HLS approach simplifies development and accelerates prototyping, its results are significantly less efficient compared to RTL designs, with notable shortcomings in both performance and resource utilization. More recently, [115] introduced a high-throughput hardware architecture for FALCON signature generation. However, a reconfigurable design supporting all processes of FALCON remains unavailable. Similarly, research on HQC hardware implementations is still scarce and relatively outdated, as HQC remains under theoretical development, with its latest specification updated in August 2025. As summarized in Table 9, only the

TABLE 9. Summary of hardware implementations for backup PQC schemes.

Ref.	Scheme	Device	Resource[1] LUT/FF/DSP/BRAM	Freq. (MHz)	Time (ms)	Novelty claims and notable characteristics
[108]	SPHINCS$^+$	Artix-7	I-s: 48.2k/72.5k/0/11.5 I-f: 48.0k/72.5k/1/11.5 III-s: 48.7k/72.5k/0/17 III-f: 48.4k/73.5k/1/17 V-s: 51.1k/74.6k/1/22.5 V-f: 51.0k/74.5k/1/22.5	250	-/12.4/0.07 - /1.01/0.16 - /21.4/0.1 - /1.17/0.19 - /19.3/0.14 - /2.52/0.21	Proposed the first hardware implementations for SPHINCS$^+$. This work introduces six designs for six SPHINCS$^+$-SHAKE variants, corresponding to 3 security levels, as well as parameter sets optimized for small and fast signature generation. The study also analyzes fault attacks on the SPHINCS$^+$ scheme and its FPGA implementation, and proposes corresponding countermeasures.
[109]	SPHINCS$^+$	ZUS+	I-s: 6.1k/4.7k/0/0.5 I-f: 6.1k/4.9k/0/0.5 V-s: 8.6k/6.4k/0/0.5 V-f: 8.6k/6.3k/0/0.5	154 156 149 152	65.8/985/1.12 2.02/64.34/2.51 131.5/1,735/2.46 8.1/199.2/4.55	Present a compact FPGA hardware implementation of the Round 3 SPHINCS$^+$. Proposed four implementations for the SHA-256 variants at security levels I and V.
[111]	SLH-DSA	Artix-7	I-s: 10.3k/7.4k/0/8 I-f: 10.2k/7.4k/0/8 III-s: 10.64k/7.64k/0/8 III-f: 10.33k/7.6k/0/8 V-s: 10.8k/7.7k/0/8 V-f: 10.4k/7.67k/0/8	150	- /363/0.64 - /19.3/1.91 - /631.6/0.92 - /31.11/2.76 - /557/1.36 - /61.77/2.81	Proposed the first hardware implementations of SLH-DSA for all parameter sets, including implementations for each security level, small and fast signature generations and both SHAKE, SHA-2 variants. The hardware resource figures in this table correspond to implementations using SHAKE as the hash function; detailed results for the SHA-2 variants are listed in [111].
[112]	SPHINCS$^+$	ZUS+	29.2/14.1/0/4	325	45/1,074/85 (μs) 0.066/1.76/0.125 0.175/3.642/0.132	Proposed a configurable SPHINCS$^+$ architecture, supporting all three processes for the SHAKE-f parameter set across three security levels.
[113]	FALCON	VUS+	I: 14.3k/7.3k/4/2 V: 13.7k/6.8k/4/2	314	- / -/7.64 (μs) - / -/14.93 (μs)	Proposed hardware architecture for FALCON verification. Introduced an efficient timing diagram for FALCON verification process.
[114]	FALCON	ZUS+	V-K: 101k/91k/1,215/69 V-S: 45.2k/41.4k/182/37 V-V: 13.3k/8.6k/15/14	100 188 214	320.3/ -/ - - /8.7/ - - / -/1.3	Proposed HLS-FPGA architectures for each of FALCON's main algorithms, supporting both security levels I and V.
[115]	FALCON	ZUS+	I-S: 80.5k/46.8k/220/45 V-S: 80.5k/46.5k/220/58	185	- / 0.865/ - - / 1.73/ -	Proposed a high-throughput architecture for FALCON signature generation on FPGA and ASIC (TSMC 28nm).
[116]	HQC	Artix-7	I-LW: 8.9k/6.4k/0/28 I-HS: 20.2k/16.4k/0/25	132 148	5.01/11.85/17.21 0.27/0.59/1.27	Proposed HLS-FPGA implementations (lightweight & high-speed) for HQC-128, developed by HQC authors alongside submission.
[119]	HQC	Artix-7	V-K: 6.8k/5.5k/0/20 V-E: 7.4k/6.6k/0/20 V-D: 18.3k/11k/0/27.5	201 187 151	138/ -/ - (μs) -/313/ - (μs) -/ -/570 (μs)	Proposed RTL-FPGA implementations for each main algorithm and security level of the HQC parameter sets, along with their operation schedules.
[120]	HQC	Artix-7	26.6k/13.6k/0/28	143	39/82/128 (μs) 96/200/305 (μs) 181/378/574 (μs)	Proposed the first unified RTL-FPGA implementation supporting all three security levels of the round 3 HQC parameter sets and all three main algorithms.

[1] I, III, V: security level; s: small signatures; f: fast signature generation; K: Keygen; S: Sign; V: Verify; LW: Lightweight; HS: High-speed.

notable work in [120] proposed a unified accelerator architecture for HQC, which supports all three KEM operations and allows runtime selection of NIST security levels, providing an important step toward flexible hardware designs.

4) Hardware optimization proposals

As mentioned above, unlike fixed software architectures, hardware architectures can be flexibly programmed and tailored to user requirements. Consequently, several hardware optimization techniques have been introduced to improve the efficiency of PQC schemes when adapted to hardware platforms. To the best of our knowledge, we summarize below a set of proposals that have been applied in PQC hardware designs and have demonstrated significant efficiency gains.

Hardware-friendly optimization design: Hardware architectures enable designers to directly interface with variables at the bit level. This flexibility allows sub-algorithms specified in PQC standards to be redesigned into hardware-friendly forms, significantly reducing resource usage and improving circuit latency. For example, Montgomery multiplication—commonly used for modular arithmetic in ML-KEM and ML-DSA, and described in detail in the ML-DSA specification [5]—introduces unnecessary complexity in hardware. Therefore, it is often replaced with Barrett reduction, as demonstrated in [61] for Kyber and [65] for Dilithium. Another approach is recursive reduction, which is specifically tailored for each modulus and leverages bit-level hardware interfaces, as described in [63] for Kyber and [82] for Dilithium. Even seemingly simple operations, such

as modular addition and subtraction, can be redesigned to better suit hardware architectures, as proposed in [94]. More recently, Jung et al. [101] introduced hardware-friendly compression algorithms for ML-KEM, which reformulated core operations into lightweight, efficient hardware designs. In summary, hardware-friendly optimization techniques showcase how designers can apply creativity and deep architectural understanding to achieve highly efficient PQC designs.

Parallel and pipeline optimization: Parallel optimization in hardware design exploits the programmable logic fabric of FPGAs to execute multiple computations simultaneously rather than sequentially. By mapping independent operations to separate processing elements, hardware architectures can achieve significantly higher throughput and lower latency. Complementarily, pipelining is a fundamental technique in synchronous digital circuit design, where registers are inserted into the critical path to reduce combinational depth between stages. This not only increases the maximum operating frequency but also enables continuous data processing. Together, parallelism and pipelining represent the most widely adopted optimization methods in PQC hardware designs.

A representative example is the NTT, where extensive research [61]–[67] demonstrates the effectiveness of these techniques. For instance, the multi-radix-2 butterfly units NTT architecture in [63] deploys 16 BUs in parallel, yielding a 16× speedup in execution. High-radix NTT designs proposed in [65] and [66] further combine parallel and pipelined structures to achieve substantial performance gains. Parallel architectures are also explored in other performance-critical modules. For example, [80] introduces a high-performance Dilithium implementation that employs three parallel Keccak cores to accelerate hashing and sampling, along with two arithmetic cores to reduce signing latency. Similarly, [87] proposes a compact architecture featuring a single Keccak core with dual permutation units, which significantly reduces sampling latency while lowering resource utilization. Fully pipelined NTT architectures, as described in [61] and [62], stand out as notable designs that restructure the complex NTT algorithm into long pipeline stages. This approach not only improves circuit frequency but also reduces memory access requirements by enabling continuous streaming of intermediate results. Accordingly, parallelism and pipelining collectively provide a powerful foundation for enhancing both performance and efficiency in PQC hardware accelerators.

Reconfigurable design and resource optimization: Leveraging reconfigurable and versatile hardware design is an effective method to exploit the flexibility of programmable architectures for processes with similar structures. In PQC hardware proposals, this optimization has been widely adopted, offering significant improvements in area efficiency and enabling multiple algorithmic processes to be supported on a single device—thereby reducing the need for multiple dedicated implementations and lowering overall cost. Several proposals focus on resource sharing at the submodule level. For example, unified arithmetic modules capable of supporting all major arithmetic operations—including both NTT and INTT—have been demonstrated in [65] for Dilithium, [63] and [89] for Kyber, and [103] for a fully unified Dilithium and Kyber implementation. Similarly, reconfigurable hash cores have been developed to support multiple SHA3 variants within a single hardware block [67], while encoder/decoder modules capable of handling all Kyber variable types were proposed in [89]. Resource optimization has also been applied to memory organization; for instance, [80] proposed grouping three BRAMs to store four Dilithium coefficients, thereby reducing memory usage by 25%. Beyond submodules, system-level reconfigurability has been explored to support all phases of PQC schemes and multiple security levels. Notable examples include configurable hardware implementations of Dilithium [85]–[88], SHAKE256-based SPHINCS$^+$ [112], Kyber [94]–[101] and HQC [120]. These designs highlight the importance of balancing efficiency with flexibility, ensuring that a single hardware platform can adapt to evolving cryptographic standards without requiring complete redesign.

Timing schedule optimization: Another key optimization strategy involves tailoring the timing schedule of hardware processes to the procedural order of PQC algorithms and the characteristics of their sub-operations. By carefully orchestrating execution order, designers can maximize submodule utilization while ensuring continuous progress along the critical computational path. Typically, this involves partitioning algorithms into multiple segments and processing each segment in a pipelined fashion. Notable examples include the detailed timing diagrams presented in [87] for ML-DSA, [101] for ML-KEM, and [120] for HQC. In [87] and [101], hashing and sampling operations are effectively overlapped with other computational processes, reducing idle cycles and boosting overall execution speed. This scheduling approach not only increases throughput and ensures high submodule utilization but also contributes to side-channel attack resistance. In particular, some SCAs—such as electromagnetic and power analysis—focus on specific processes (e.g., modular reduction steps like Barrett reduction) to exploit distinctive leakage patterns [123]–[129]. By overlapping these sensitive processes with other computations and minimizing predictable idle windows, optimized schedules reduce the visibility of such targets, thereby enhancing security against process-specific leakage exploitation.

D. PQC IMPLEMENTATIONS ON ASIC

An application-specific integrated circuit (ASIC) is a specialized integrated circuit (IC) chip designed for a dedicated function, in contrast to general-purpose processors. Typically fabricated using metal–oxide–semiconductor (MOS) technology, ASICs are optimized at the hardware level to achieve superior performance, power efficiency, and area utilization for their target application. In the field of PQC, ASICs offer a compelling platform to accelerate core operations such as polynomial multiplications, modular reductions, and lattice-based sampling, which are often computational bottlenecks. Their tailored architecture enables not only high-throughput cryptographic processing but also significant energy savings,

TABLE 10. Summary of hardware implementations for PQC schemes in ASIC.

Ref.	Scheme	Tech.	L. Gates (kGE)	Memory (KB)	Freq. (MHz)	Clock cycle (kCCs)	T. time (μs)	Energy (μJ)	Novelty claimns and notable characteristics
[134]	Kyber	28 nm TSMC	942 + 37*	12	300	I: 18.6/45.9/80.0 V: 39.7/81.6/136.5	482 859	12.8 24.26	Propose a configurable cryptographic processor for Kyber on a RISC-V platform and post-synthesis implemented under TSMC 28nm CMOS technology.
[136]	Kyber	40 nm	532	14	80-115	I: 12.245 (total K+E+D)	-	2.76	Propose an ASIC design manufactured at 40nm, with a total chip core area of only 0.6mm^2.
[138]	Kyber	40 nm	253	7	10-270	I: 21.41 (total K+E+D)	-	18.5	Propose a fabricated chip using a 40nm LP CMOS process, supporting the Kyber-512 scheme.
[139]	Kyber & Dilithium	28nm	581+116*	24.75	540	I: 9.4/19/43.8 III: 14.2/26.2/59.1 V: 18.5/33.7/77.5 II: 48.5/175.1/89.8 III: 68.4/224.6/110 V: 94.9/313.2/160	133.7 184.3 536.5 575.4 746.9 1052	11.11 14.93 19.19 106 62.39 88.35	Propose an HW/SW co-design based on RISC-V, implemented under TSMC 28nm CMOS technology, supporting Kyber and Dilithium.
[103]	Kyber & Dilithium	28 nm	747	34.82	1000	V: 25.26 (E+D) III: 73.55 (S+Ve)	-	9.27 27	Propose a configurable implementation of Kyber and Dilithium on 65nm and 28 nm technologies.

* RISC-V core resource; K: Keygen; E: Encapsulation; D: Decapsulation; S: Signature generation; Ve: Verification.

making them highly suitable for both large-scale cloud infrastructures and resource-constrained devices such as IoT and mobile platforms. As a result, ASIC-based PQC implementations provide a practical pathway toward secure, efficient, and scalable deployment of quantum-resistant cryptographic schemes in real-world applications.

In academic settings, several hardware/software co-designs have emerged to support high-performance implementations of PQC algorithms. Banerjee et al. [133] prototype a RISC-V microprocessor tightly coupled with a Sapphire custom crypto core, fabricated in TSMC's 40 nm low-power CMOS process, supporting both Kyber and Dilithium (NIST Round 2 finalists). In related work, other teams have developed RISC-V-based designs using TSMC 28 nm HPC+ technology (0.9 V) [134], [139], evaluated via post-synthesis implementation using Synopsys Design Compiler—details of which are summarized in Table 10. Aikata et al. [102] further present a post-synthesis configurable ASIC for Kyber and Dilithium, while other groups have fabricated dedicated Kyber implementations in 40 nm technology [136], [138]. For hash-based PQC, Karl et al. [140] propose an ASIC SHA-3 core in 22 nm, accelerating hash functions used across NIST candidates. More recently, research [141] explores a hardware/software co-design targeting the SHAKE-based SLH-DSA, implemented on a customized RISC-V core and evaluated in 28 nm technology. At present, ASIC implementations for FALCON and HQC remain unpublished or not yet fabricated.

Recognizing the critical importance of PQC for future communications standards and security systems, several commercial efforts are now pioneering real silicon solutions. In September 2024, the global team at PQShield—consisting of leading post-quantum cryptographers, hardware designers, and engineers—announced the first PQC-compliant silicon test chip, enabling real-world validation of performance, power consumption, and side-channel resilience [142]. SK Telecom, in collaboration with cryptographic hardware supplier KCS, has introduced a commercial quantum encryption chip named "Q-HSM" [143]. In 2025, Samsung Electronics released the S3SSE2A, billed as the industry's first security chip with embedded hardware PQC support, intended for integration into future mobile devices [144]. In the same year, UK chip designer EnSilica developed a combined PQC hardware accelerator IP block, integrating Dilithium, Kyber and SHA-3 into a single solution to reduce silicon area, power, and cost [145]. SEALSQ Corp has also completed testing and initiated the certification process for its QS7001 Post-Quantum Secure Chip, scheduled for release in 2025. This chip strengthens blockchain ecosystems against quantum threats using NIST-compliant PQC algorithms, including ML-KEM and ML-DSA [146]. Collectively, these developments highlight the growing industrial commitment to practical and secure deployment of PQC across diverse platforms.

IV. SIDE-CHANNEL ATTACKS AND COUNTERMEASURE METHODS FOR PQC IMPLEMENTATIONS

SCAs remain a persistent threat to all cryptographic schemes—including PQC algorithms—because they exploit information unintentionally leaked by a system through timing behavior, power consumption, and electromagnetic emissions. Such leakages can reveal sensitive information even when the underlying cryptographic algorithm is mathematically secure. Therefore, SCA resilience must be carefully considered in PQC implementations, with countermeasures tailored to the specific hardware/software platform and to the structural properties of each cryptographic scheme. Building on the comprehensive studies and surveys on SCAs presented in [121], [122], along with several notable SCA demonstrations and countermeasures for PQC schemes in [123]–[131],

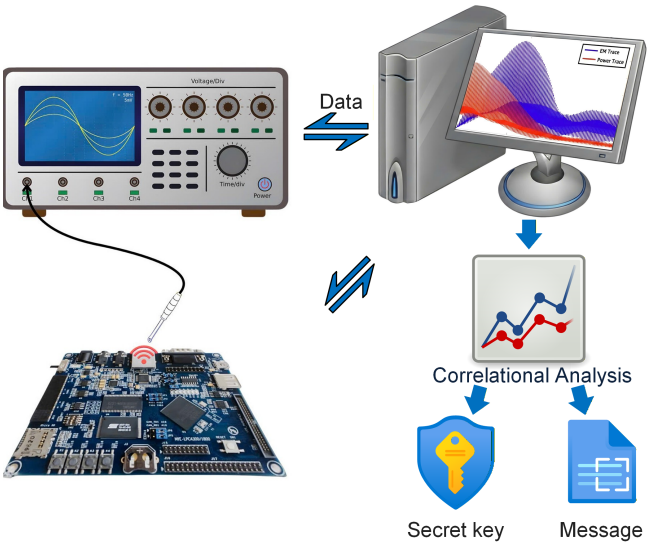

FIGURE 12. Overview of the side-channel attacks setup.

the general classes of SCAs relevant to PQC include:

Timing attacks: These attacks exploit variations in execution time to infer sensitive information and have already been demonstrated against modern cryptographic standards [123].

Power analysis: These attacks leverage variations in power consumption—through single, differential, or correlation power analysis—to extract secret information from hardware during computation.

Electromagnetic (EM) analysis: These attacks involve capturing EM radiation emitted by a device during operation to infer sensitive data.

Cache and memory attacks: Exploit timing or access patterns in shared memory hierarchies (L1/L2 cache, DRAM) to infer secret data processed by another program. By observing cache hits/misses, evictions, or memory row behavior, attackers can recover cryptographic keys without direct access.

Fault injection attacks: These involve deliberately inducing abnormal conditions in a processor, chip, or cryptographic module during execution. For PQC implementations, the comprehensive survey in [132] provides detailed analyses of fault targets within the mathematical structures of both primary NIST PQC algorithms.

Besides the most typical and notable SCAs listed above, many additional side-channel vectors can be applied to compromise PQC schemes across different hardware and software platforms. To the best of our knowledge, several comprehensive studies in this field have been introduced in [123]–[131]. Among these, power and EM analysis remain the most widely used techniques, and an overview of a typical attack setup is illustrated in Fig. 12. The study in [124] presents power analysis attacks exploiting leakage from the Barrett reduction in Kyber. Power and EM analysis have also been used in [125], [126] to target the Kyber decryption process. For DSA implementations, Karabulut et al. [127] proposed power-trace attacks on the sampling process of Dilithium, while Steffen et al. [128] demonstrated correlation power analysis on the NTT and multiplication operations of Dilithium. Recently, [129] introduced fault injection attacks targeting Keccak, the hash function used in both primary NIST PQC schemes. For the backup PQC candidates, the work in [130] proposed a power-based attack strategy against the Reed–Muller and Reed–Solomon components of HQC cryptosystems.

Recognizing the threat that SCAs pose to cryptosystems—including PQC—numerous countermeasures are proposed, many of which are evaluated within the same studies cited above. These countermeasures are built not only on the mathematical security properties of each PQC algorithm but also on the characteristics of the implementation platform. Therefore, for any specific application, researchers must carefully analyze and select appropriate protection techniques. Due to the inherently physical nature of SCAs, PQC applications must follow device- and channel-level security guidelines recommended by NIST for DSA and KEM schemes [16], [27]. In addition, several typical and effective advanced countermeasures introduced in [123]–[131] include:

Constant-time design: A highly effective method against timing attacks and applicable to all PQC designs. For example, timing-attack-resistant designs are presented in [132].

Masking: Splits every sensitive intermediate value into multiple random shares such that each share alone reveals no information about the secret. This technique is widely used in KEM and DSA, as demonstrated in [125], [126] and [128].

Hiding: Conceals sensitive operations by injecting noise or overlapping them with other processes. For instance, [128] leverages hashing and sampling to obscure leakage from Dilithium's arithmetic operations.

Dual-Rail (balanced) logic : Balances power consumption between complementary signal paths to achieve power traces that are independent of processed data.

Decoupling: Electrically isolates the cryptographic core from the system power supply so that external measurements cannot directly observe its instantaneous power variations.

V. APPLICATIONS OF POST-QUANTUM CRYPTOGRAPHY

Currently, cryptographically relevant quantum computers do not yet exist, and the standardization-to-deployment cycle for PQC is expected to span roughly a decade. This extended timeline arises because enterprises, governments, and service providers must carefully integrate new algorithms into widely used products, protocols, and infrastructures [147]. Building on the roles of today's DSA and KEM standards, PQC is envisioned to replace vulnerable cryptographic primitives and safeguard critical applications across diverse domains.

Secure Communications: PQC standards are poised to replace current public-key algorithms in widely deployed protocols such as Transport Layer Security (TLS), Internet Protocol Security (IPsec), and Virtual Private Networks (VPNs). These protocols underpin secure web browsing (HTTPS), enterprise VPNs, email encryption, and secure messaging applications [148]. In addition, PQC will be indispensable for securing next-generation 5G/6G networks and IoT device-

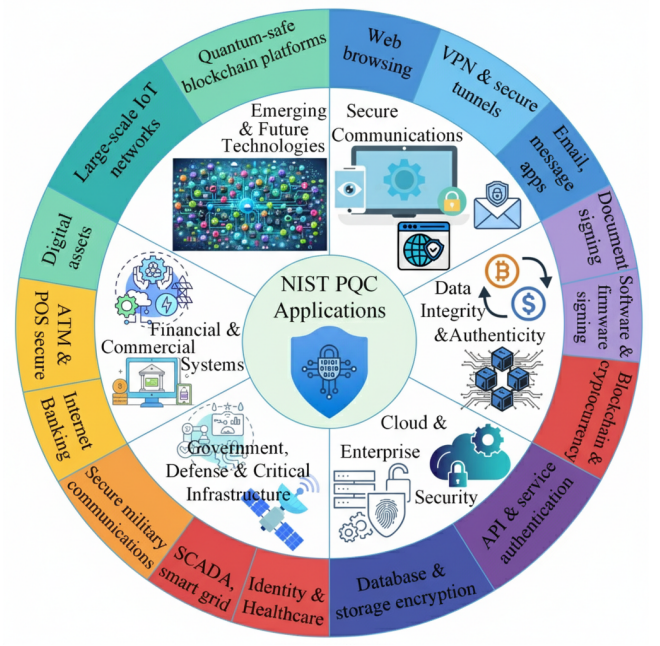

FIGURE 13. **Applications of post-quantum cryptography.**

to-cloud communication links, where confidentiality and authenticity are fundamental.

Data Integrity and Authenticity: PQC-based digital signatures will play a crucial role in verifying the integrity and authenticity of digital assets. Applications include document signing, software code signing, and secure updates for embedded devices. Blockchain systems are also a promising use case [149], with Komodo [150] reporting the adoption of a modified Dilithium signature scheme, while Tidecoin has explored FALCON for quantum-secure crypto transactions [151]. In enterprise workflows, PQC signatures will help maintain trusted audit trails, prevent tampering, and support compliance with stringent regulatory frameworks in finance, healthcare, and government sectors.

Cloud, Web, and Enterprise Security: Large-scale data centers, enterprise platforms, and Software as a Service (SaaS) applications depend on robust cryptographic foundations. PQC will replace vulnerable algorithms in identity and access management, Single Sign-On (SSO), and data-at-rest key management. By transitioning to PQC, enterprises can ensure long-term confidentiality of customer data, mitigate quantum threats, and maintain compliance with data protection regulations across finance, e-commerce, and healthcare.

Financial and Commercial Systems: The financial ecosystem—including digital banking, blockchain systems and mobile payment platforms—relies on both encryption and digital signatures for secure transaction verification. PQC strengthens resilience against quantum-enabled attacks on critical infrastructures such as EMV payment standards, SWIFT communications and cryptocurrency wallets. By adopting PQC, financial institutions can safeguard customer trust and reduce systemic risks to the global financial system.

Government, Defense and Critical Infrastructure: Governments and defense agencies require quantum-safe cryptography to secure national security assets, military command systems, and satellite communication. PQC provides robust protection for sensitive data, sensor networks, and classified communication channels. Beyond defense, civilian critical infrastructures—such as power grids, transportation systems, and healthcare networks—will also integrate PQC to mitigate emerging quantum-era threats.

Emerging and Future Applications: Looking ahead, PQC enables resilience in new areas such as secure software updates, trusted firmware distribution, and edge computing deployments. These applications are particularly critical for IoT and automotive systems, where compromise could lead to catastrophic outcomes. Hybrid cryptographic deployments—combining classical and PQC algorithms—are already under exploration, paving the way for a smooth transition to fully quantum-safe infrastructures.

VI. CONCLUSION

As the specter of quantum computing advances, PQC algorithms have become essential for safeguarding modern digital systems—ranging from secure communications and financial platforms to critical infrastructure and data privacy. Their resilience against emerging quantum threats makes them crucial for preserving trust, integrity, and confidentiality in real-world applications. Beyond securing communications, PQC also plays a vital role in ensuring long-term data protection and securing cloud services, IoT networks, 5G infrastructures, and even blockchain systems against future quantum-enabled attacks. In this tutorial review, we have provided a comprehensive overview of PQC algorithms by introducing their core concepts and presenting consistent, simple, and illustrative toy examples across different algorithms to enhance conceptual understanding. We highlighted both the theoretical foundations and their practical significance. By organizing and clarifying the underlying principles, we aim to make the field more accessible to researchers. Furthermore, we surveyed the current landscape of software and hardware implementations—spanning general-purpose processors, GPUs, FPGAs, and embedded systems—demonstrating how PQC is progressing from theoretical design toward practical deployment across diverse platforms.

REFERENCES

[1] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1484–1509, Oct. 1997.
[2] NIST, "Submission requirements and evaluation criteria for the post-quantum cryptography standardization process," Available at: https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf
[3] NIST, "Status report on the third round of the NIST-PQC standardization Process,". Accessed: July 2022. [Online]. Available at: https://csrc.nist.gov/projects/post-quantum-cryptography.
[4] NIST, "FIPS 203- Module-Lattice-Based Key-Encapsulation Mechanism Standard," accessed: Aug. 2024. [Online]. Available at: https://csrc.nist.gov/pubs/fips/203/final

[5] NIST, "FIPS 204- Module-Lattice-BasedDigital Signature Standard," accessed: Aug. 2024. [Online]. Available at: https://csrc.nist.gov/pubs/fips/204/final

[6] NIST, "FIPS 205- Stateless Hash-Based Digital Signature Standard," accessed: Aug. 2024. [Online]. Available at: https://csrc.nist.gov/pubs/fips/205/final

[7] FALCON: Fast-fourier lattice-based compact signatures over NTRU. Specification v1.2. Accessed: Oct. 1, 2020. [Online]. Available at: https://falcon-sign.info/

[8] P. Gaborit et al., "Hamming Quasi-Cyclic (HQC)". Accessed: Aug. 2025. [Online]. Available at: https://pqc-hqc.org/doc/hqc_specifications_2025_08_22.pdf.

[9] NIST, "Status report on the fourth round of the NIST-PQC standardization Process,". Accessed: Mar. 2025. [Online]. Available at: https://csrc.nist.gov/projects/post-quantum-cryptography.

[10] D.-T. Dam et al., "A survey of post-quantum cryptography: Start of a new race," *Cryptography*, vol. 7, no. 3, p. 40, Aug. 2023.

[11] J. Xie, W. Zhao, H. Lee, D. B. Roy and X. Zhang, "Hardware Circuits and Systems Design for Post-Quantum Cryptography—A Tutorial Brief," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 71, no. 3, pp. 1670-1676, Mar. 2024.

[12] A. Wang, D. Xiao, and Y. Yu, "Lattice-based cryptosystems in standardisation processes: A survey," *IET Information Security*, vol. 17, no. 2, pp. 227–243, Mar. 2023.

[13] H. Nguyen, S. Huda, Y. Nogami and T. T. Nguyen, "Security in Post-Quantum Era: A Comprehensive Survey on Lattice-Based Algorithms," *IEEE Access*, vol. 13, pp. 89003-89024, 2025.

[14] S. Ahmadunnisa and S. E. Mathe, "A comprehensive review on hardware implementations of lattice-based cryptographic schemes," *Journal of Systems Architecture*, vol. 167, 2025.

[15] NIST (2023), Digital Signature Standard (DSS).(Department of Commerce, Washington, D.C.), Federal Information Processing Standards Publication NIST FIPS 186-5. Available at: https://doi.org/10.6028/NIST.FIPS.186-5

[16] E. Barker, "Recommendation for obtaining assurances for digital signature applications," NIST Special Publication (SP) 800-89. Available at: https://doi.org/10.6028/NIST.SP.800-89.

[17] E. Kiltz, V. Lyubashevsky and C. Schaffner, "A concrete treatment of Fiat–Shamir signatures in the quantum random-oracle model," in *Proc.37th Annu. Int. Conf. Theory Appl. Cryptograph. Techn.*, pp. 552–586, Tel Aviv, Israel, Apr. 2018.

[18] Open quantum safe (OQS) algorithm performance visualizations. Available at https://openquantumsafe.org/benchmarking.

[19] Vadim Lyubashevsky, "Fiat-shamir with aborts: Applications to lattice and factoring-based signatures," in *International Conference on the Theory and Application of Cryptology and Information Security*, Springer Berlin Heidelberg, pp. 598–616, 2009.

[20] NIST, "FIPS 202- SHA3 Standard: Permutation-Based Hash and Extendable-Out Function," Accessed: Aug. 2015. [Online]. Available at: https://csrc.nist.gov/pubs/fips/202/final

[21] S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehle, "CRYSTALS-Dilithium," Proposal to NIST PQC Standardization, Round3, 2021, https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions.

[22] J-P. Aumasson et al., "SPHINCS$^+$ Submission to the NIST post-quantum project, v.3.1," June, 2022, https://sphincs.org/data/sphincs+-r3.1-specification.pdf.

[23] NIST, "FIPS PUB 180-4: Secure Hash Standard (SHS)," accessed: Aug. 2015. [Online]. Available at https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf.

[24] E. Barker et al., "Recommendation for random bit generator (RBG) constructions," NIST Special Publication 800-90C 4pd, 2024. Available at: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90C.4pd.pdf

[25] D. Boneh et al., "Random Oracles in a Quantum World," in *Advances in Cryptology – ASIACRYPT 2011*, Lecture Notes in Computer Science, vol. 7073, Springer, Berlin, Heidelberg.

[26] C. Gentry, C. Peikert and V. Vaikuntanathan, "Trapdoors for hard lattices and new cryptographic constructions," in *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, pp. 197-206, USA, 2008.

[27] NIST, "Recommendations for Key-Encapsulation Mechanisms," NIST Special Publication (SP) 800-227. Accessed: Jan. 2025. [Online]. Forthcoming; will be available at: https://csrc.nist.gov/publications

[28] E. Barker et al., "Recommendation for pair-wise key-establishment schemes using discrete logarithm cryptography," NIST Special Publication 800-56A Revision 3, 2018. Available at: https://doi.org/10.6028/NIST.SP.800-56Ar3

[29] E. Barker et al., "Recommendation for pair-wise key-establishment using integer factorization cryptography," NIST Special Publication 800-56B Revision 2, 2019. Available at: https://doi.org/10.6028/NIST.SP.800-56Br2

[30] NIST, "FIPS 197: Advanced Encryption Standard (AES)," accessed: May 2023. [Online]. Available at https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf.

[31] R. Avanzi, "CRYSTALS-Kyber algorithm specifications and supporting documentation," version 3.02, Aug. 2021. Available at https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf.

[32] E. Fujisaki and T. Okamoto, "Secure Integration of Asymmetric and Symmetric Encryption Schemes," *Journal of Cryptology*, vol. 26, pp. 80-101, 2013.

[33] R. Gonzalez, "Kyber - How does it work?". [Online]. Available at: https://cryptopedia.dev/posts/kyber.

[34] C. Aguilar-Melchor, O. Blazy, J. -C. Deneuville, P. Gaborit and G. Zémor, "Efficient Encryption From Random Quasi-Cyclic Codes," *IEEE Transactions on Information Theory*, vol. 64, issue 5, pp. 3927-3943, May. 2018.

[35] D. Hofheinz, K. Hövelmanns and E. Kiltz, "A Modular Analysis of the Fujisaki-Okamoto Transformation," in *Theory of Cryptography Conference*, Springer, vol. 10677, pp. 341-371, 2017.

[36] S. Shen, H. Yang, W. Dai, H. Zhang, Z. Liu, and Y. Zhao, "High throughput GPU implementation of dilithium post-quantum digital signature," *IEEE Trans. Parallel Distrib. Syst.*, vol. 35, no. 11, pp. 1964–1976, Nov. 2024.

[37] S. Shen, H. Yang, W. Li and Y. Zhao, "cuML-DSA: Optimized Signing Procedure and Server-Oriented GPU Design for ML-DSA," *IEEE Trans. on Depen.and Sec. Computing*, vol. 22, no. 3, pp. 2295-2307, 2025.

[38] S. C. Seo et al., "Accelerating Falcon Post-Quantum Digital Signature Algorithm on Graphic Processing Units," *Computers, Materials and Continua*, vol. 75, no.1, pp. 1963-1980, Jan. 2023.

[39] W. -K. Lee et al., "High Throughput Lattice-Based Signatures on GPUs: Comparing Falcon and Mitaka," *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 4, pp. 675-692, April 2024.

[40] W. Li et al., "cuFalcon: An Adaptive Parallel GPU Implementation for High-Performance Falcon Acceleration," *Cryptology ePrint Archive*, paper 2025/249, Feb. 2025.

[41] R. Dai et al., "GOLF: Unleashing GPU-Driven Acceleration for FALCON Post-Quantum Cryptography," *Cryptology ePrint Archive*, paper 2025/749, Apr. 2025.

[42] X. Ji et al., "HI-Kyber: A Novel High-Performance Implementation Scheme of Kyber Based on GPU," *IEEE Trans. on Parallel and Distributed Systems*, vol. 35, no. 6, pp. 877-891, 2024.

[43] Z. Wang, X. Dong, H. Chen, Y. Kang and Q. Wang, "CUSPX: Efficient GPU Implementations of Post-Quantum Signature SPHINCS+," *IEEE Transactions on Computers*, vol. 74, no. 1, pp. 15-28, Jan. 2025.

[44] D. Kim, H. Choi and S. C. Seo, "Parallel Implementation of SPHINCS+ With GPUs," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 71, no. 6, pp. 2810-2823, June 2024.

[45] J. Wu et al., "CBPSPX: A CUDA-based Batch Parallel Optimization of Post-Quantum Signature SPHINCS+," *IEEE Internet of Things Journal*, vol. 12, no. 18, pp. 37898-37911, Sep. 2025.

[46] N. Gupta, A. Jati, A. K. Chauhan, and A. Chattopadhyay, "PQC acceleration using GPUs: FrodoKEM, NewHope, and Kyber," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 3, pp. 575–586, Mar. 2021.

[47] M. J. Kannwischer et al., "pqm4: Benchmarking NIST Additional Post-Quantum Signature Schemes on Microcontrollers," *Cryptology ePrint Archive*, paper 2024/112, Jan. 2024.

[48] S. C. Seo, S. W. An, "Parallel implementation of CRYSTALS-Dilithium for effective signing and verification in autonomous driving environment," *ICT Express*, vol. 9, issue 1, pp. 100–105, Feb. 2023.

[49] J. Huang et al., "Improved plantard arithmetic for lattice-based cryptography," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 1, pp. 614–636, Aug. 2022.

[50] H. Becker, V. Hwang, M. J. Kannwischer, B.-Y. Yang, and S.-Y. Yang, "Neon NTT: Faster Dilithium, Kyber, and Saber on Cortex-A72 and Apple M1," *IACR Transactions on Cryptographic Hardware and Embedded Systems* vol. 2022, no. 1, pp. 221-244, 2021.

[51] A. Abdulrahman, V. Hwang, M. J. Kannwischer, and A. Sprenkels, "Faster Kyber and Dilithium on the Cortex-M4," in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur*. Rome, Italy: Springer, Jun. 2022, pp. 853–871.

[52] Y. Kim, S. Yoon and S. C. Seo, "Vectorized Implementation of Kyber and Dilithium on 32-bit Cortex-A Series," *IEEE Access*, vol. 12, pp. 104414-104428, 2024.

[53] P. Sanal, E. Karagoz, H. Seo, R. Azarderakhsh, and M. Mozaffari-Kermani, "Kyber on ARM64: Compact implementations of Kyber on 64-bit arm cortex-a processors," in *Proc. Int. Conf. Secur.Privacy Commun. Syst*. Cham, Switzerland: Springer, Sep. 2021, pp. 424–440.

[54] S. Aghapour et al., "PUF-Kyber: Design of a PUF-Based Kyber Architecture Benchmarked on Diverse ARM Processors," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 12, pp. 4453-4462, Dec. 2024.

[55] S. Aghapour et al., "PUF-Dilithium: Design of a PUF-Based Dilithium Architecture Benchmarked on ARM Processors," *ACM Trans. on Embedded Computing Systems*, vol. 24, issue. 2, pp. 1-20, Feb. 2025.

[56] Y. Kim, J. Song and S. C. Seo, "Accelerating Falcon on ARMv8," *IEEE Access*, vol. 10, pp. 44446-44460, 2022.

[57] M. A. Sayed and M. Taha, "A High Speed Post-Quantum Digital Signature at 180 Sig/Sec On ARM Cortex-M4," in *2023 8th Inter. Conf. on Multimedia Communication Technologies (ICMCT)*, China, 2023, pp. 43-49.

[58] Mupq, "Mupq/pqm4: Post-quantum crypto library for the arm cortex-m4". [Online]. Available: https://github.com/mupq/pqm4.

[59] D. Kim, J. Choi, S. Yoon and S. C. Seo, "Optimized implementation of HQC on Cortex-M4," *ICT Express*, vol.11, no. 5, pp. 939-944, Oct. 2025.

[60] P. Duong-Ngoc et al., "Area-Efficient Number Theoretic Transform Architecture for Homomorphic Encryption," *IEEE Trans. Circuits and Systems I, Regular Papers*, vol. 70, no. 6, pp. 1270-1283, Mar. 2023.

[61] P. Duong-Ngoc and H. Lee, "Configurable Mixed-Radix Number Theoretic Transform Architecture for Lattice-Based Cryptography," *IEEE Access*, vol. 10, pp. 12732-12741, 2022.

[62] T. H. Nguyen, B. Kieu-Do-Nguyen, C. K. Pham and T. T. Hoang, "High-speed NTT Accelerator for CRYSTAL-Kyber and CRYSTAL-Dilithium," *IEEE Access*, vol. 12, pp. 34918-34930, 2024.

[63] F. Yaman et al., "A hardware accelerator for polynomial multiplication operation of CRYSTALS-KYBER PQC scheme," in *2021 Design, Automation Test in Europe Conference Exhibition*, pp. 1020-1025. IEEE.

[64] K. Javeed and D. Gregg, "Efficient Number Theoretic Transform Architecture for CRYSTALS-Kyber," *IEEE Trans. Circuits Syst. II*, vol. 72, no. 1, pp. 263-267, Jan. 2025.

[65] T.X. Pham, P. Duong-Ngoc and H. Lee, "An efficient unified polynomial arithmetic unit for CRYSTALS-Dilithium," *IEEE Trans. Circuits Syst. 1*, vol. 70, no. 12, pp. 4854-4864, Dec. 2023.

[66] S. Mandal and D. B. Roy, "Winograd for NTT: A Case Study on Higher-Radix and Low-Latency Implementation of NTT for Post Quantum Cryptography on FPGA," *IEEE Trans. Circuits and Systems I, Regular Papers*, vol. 71, no. 12, pp. 6396-6409, Dec. 2024.

[67] Q. D. Truong and H. Lee, "Efficient polynomial arithmetic and hash modules for ML-DSA and ML-KEM standards," in 2024 *IEEE Asia Pacific Conf. on Circuits and Systems*, Taiwan, 2024, pp. 776-780.

[68] T-H. Nguyen et al. "A Low-Latency Polynomial Arithmetic Unit for ML-KEM and ML-DSA Standards," in *2025 IEEE International Symposium on Circuits and Systems (ISCAS)*, London, United Kingdom, May 2025.

[69] Q. D. Truong, P. N. Duong and H. Lee, "Hybrid number theoretic transform architecture for homomorphic encryption," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 33, issue. 7, Jul. 2025.

[70] G. Alsuhli, H. Saleh, M. Al-Qutayri, B. Mohammad and T. Stouraitis, "Area and Power Efficient FFT/IFFT Processor for FALCON PostQuantum Cryptography," *IEEE Trans. on Emerging Topics in Computing*, vol. 13, no. 2, pp. 423-437, Apr.-Jun. 2024.

[71] DT. Dam et al. "Compact FALCON FFT/NTT Accelerator for Post-Quantum Cryptography," in *2025 IEEE International Symposium on Circuits and Systems (ISCAS)*, London, United Kingdom, May 2025.

[72] JH. Liao et al. "A Unified FFT/NTT Design for Efficient NTRU Equation Solving in FALCON Cryptography," in *2025 IEEE International Symposium on Circuits and Systems (ISCAS)*, London, UK, May 2025.

[73] Y. Tu et al., "LEAP: Lightweight and Efficient Accelerator for Sparse Polynomial Multiplication of HQC," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 6, pp. 892-896, June 2023.

[74] G. Bertoni, J. Daemen, S. Hoffert, M. Peeters, and G. V. Assche.(2020)., *Keccak in VHDL*. [Online]. Available:https://keccak.team/hardware.html

[75] CERG, *SHAKE*, https://github.com/GMUCERG/SHAKE, 2021.

[76] H. L. Pham, T. H. Tran, V. T. Duong Le and Y. Nakashima, "A High-Efficiency FPGA-Based Multimode SHA-2 Accelerator," *IEEE Access*, vol. 10, pp. 11830-11845, 2022.

[77] K. Javeed and D. Gregg, "CUBA: A Configurable Unified Butterfly Architecture for Lattice-based Cryptosystems," *IEEE Embedded Systems Letters*, May 2025 (*early access*).

[78] S. Ricci, L. Malina, P. Jedlicka, D. Smekal, 1. Hajny, P. Cibik, and P. Dobias, "Implementing CRYSTALS-Dilithium Signature Scheme on FPGAs," *Cryptology ePrint Archive* 2021/108, Jan. 2021.

[79] G. Land, P. Sasdrich, and T. Guneysu, "Hard Crystal-Implementing Dilithium on Reconfigurable Hardware," *Cryptology ePrint Archive* 2021/355, Mar. 2021.

[80] L. Beckwith, D. T. Nguyen, and K. Gaj, "High-performance hardware implementation of CRYSTALS-Dilithium," *Crypto. ePrint Arch.*, Report 2021/1451, 2021.

[81] 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, no. 1, pp. 270–295, 2022.

[82] T. Wang, C. Zhang, P. Cao and D. Gu, "Efficient implementation of Dilithium signature scheme on FPGA SoC Platform," *IEEE Trans. on Very Large Integration (VLSI) systems*, vol. 30, no. 9, pp. 1158-1171, Sep. 2022.

[83] N. Gupta, A. Jati, A. Chattopadhyay, and G. Jha, "Lightweight Hardware Accelerator for Post-Quantum Digital Signature CRYSTALS-Dilithium," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 70, no. 8, pp. 3234-3243, Aug. 2023.

[84] H. Zhao et al., "Sparse Polynomial Multiplication-Based High-Performance Hardware Implementation for CRYSTALS-Dilithium," in *2024 IEEE Inter. Symp. on Hard. Oriented Secur. and Trust (HOST)*, USA, 2024, pp. 150-159.

[85] X. Li, J. Lu, D. Liu, A. Li, S. Yang and T. Huang, "A High Speed Post-Quantum Crypto-Processor for Crystals-Dilithium," in *IEEE Trans. on Cirs. and Systems II: Express Briefs*, vol. 71, no. 1, pp. 435-439, Jan. 2024.

[86] L.Beckwith, D. T. Nguyen and K.Gaj,"HardwareAccelerators for Digital Signature Algorithms Dilithium and FALCON," *IEEE Design & Test*, vol. 41, no. 5, pp. 27-35, Oct. 2024.

[87] Q. D. Truong, P. N. Duong and H. Lee, "Efficient low-latency hardware architecture for Module-Lattice-Based Digital Signature Standard," *IEEE Access*, vol. 12, pp. 32395-32407, Feb. 2024.

[88] Y. Cui et al., "High-Performance Hardware Implementation of Crystals-Dilithium Based on Improved MDC-NTT," in *IEEE Trans. on Computers*, vol. 74, pp. 2896-2908, Sep. 2025.

[89] Y. Xing and S. Li, "A compact hardware implementation of CCA-secure key exchange mechanism CRYSTALS-KYBER on FPGA," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2021, no. 2, pp. 328–356, 2021.

[90] M. Bisheh-Niasar et al., "Instruction-set accelerated implementation of CRYSTALS-Kyber," *IEEE Trans. Circuits Syst. I: Regular Papers*, vol. 68, no. 11, pp. 4648–4659, 2021.

[91] W. Guo, S. Li, and L. Kong, "An efficient implementation of KYBER," *IEEE Trans. on Circuits and Systems II, Exp. Briefs*, vol. 69, no. 3, pp. 1562–1566, Mar. 2022.

[92] V. B. Dang, K. Mohajerani, and K. Gaj, "High-speed hardware architectures and FPGA benchmarking of CRYSTALS-Kyber, NTRU, and saber," *IEEE Trans. on Computer*, vol. 72, no. 2, pp. 306–320, Feb. 2023.

[93] Z. Ni, A. Khalid, D.-S. Kundi, M. O'Neill, and W. Liu, "HPKA: A High-Performance CRYSTALS-Kyber Accelerator Exploring Efficient Pipelining," *IEEE Trans. on Computers*, vol. 72, pp. 3340-3353, Dec. 2023.

[94] W. Guo and S. Li, "Highly-efficient hardware architecture for CRYSTALS-Kyber with a novel conflict-free memory access pattern," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 70, no. 11, pp. 4505–4515, Nov. 2023.

[95] W. Guo and S. Li, "Split-radix based compact hardware architecture for CRYSTALS-Kyber," *IEEE Transactions on Computer*, vol. 73, no. 1, pp. 97–108, Jan. 2024.

[96] J. Zhang et al., "Super-K: A Superscalar CRYSTALS-KYBER Processor Based on Efficient Arithmetic Array," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 71, no. 9, pp. 4286-4290, Sept. 2024.

[97] T-H. Nguyen et al., "Efficient hardware implementation of the lightweight CRYSTALS-Kyber," in *IEEE Trans. Circuits Syst. I*, vol. 72, no. 2, pp. 610-622, Feb. 2025.

[98] Y. Cui, J. Chen, Z. Ni, Z. Zhang, C. Wang and W. Liu, "Instruction-Based High-Performance Hardware Controller of CRYSTALS-Kyber With Balanced Resource Utilization," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 72, no. 5, pp. 2394-2407, May 2025.

[99] H. Kim et al., "A configurable ML-KEM/Kyber Key-Encapsulation hardware accelerator architecture," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 71, issue 11, pp. 4678-4682, Nov. 2024.

[100] T. H. Nguyen et al., "An Area-Time Efficient Hardware Architecture for ML-KEM Post-Quantum Cryptography Standard," *IEEE Access*, vol. 13, pp. 103834-103847, Jun. 2025.

[101] H. Jung, Q. D. Truong and H. Lee, "Highly-Efficient Hardware Architecture for ML-KEM PQC Standard," *IEEE Open Journal of Circuits and Systems*, vol. 6, pp. 356-369, 2025.

[102] A. Li', Z. Li, J. Tang and Y. Lu, "KDA: Kyber and Dilithium Accelerator for CRYSTALS Suite of Post-Quantum Cryptography in Hybrid Multipath Delay Commutator Pipelined Architecture," in *2024 IEEE Asian Solid-State Circuits Conference (A-SSCC)*, Japan, 2024, pp. 1-3.

[103] A. Aikata, C. Mert, M. Imran, S. Pagliarini and S. Roy, "KaLi: A crystal for post-quantum security using Kyber and Dilithium," *IEEE Transactions on Circuits and Systems I*, vol. 70, no. 2, pp.747-758, Feb. 2023.

[104] P. Dobias, L. Malina and J. Hajny, "Efficient unified architecture for post-quantum cryptography: combining Dilithium and Kyber," *PeerJ Computer Science*, vol. 11, Mar. 2025.

[105] Q. D. Truong et al., "High-Performance Unified Hardware Architecture for ML-DSA and ML-KEM PQC Standards," *IEEE Access*, vol. 13, pp. 189444-189460, 2025.

[106] "Automated Cryptographic Validation Test System- Gen/Vals," GitHub. Available at: https://github.com/usnistgov/ACVP-Server. [Online]. Accessed: May 2025.

[107] "Redefining Security by Enabling Post-Quantum Cryptography on Agilex 5 System on Module", *iWave Global*, available at: https://iwave-global.com/articles/post-quantum-cryptography-on-agilex-5-system-on-module/.

[108] D. Amiet, L. Leuenberger, A. Curiger and P. Zbinden, "FPGA-based SPHINCS+ Implementations: Mind the Glitch," in *2020 23rd Euromicro Conference on Digital System Design*, Slovenia, 2020, pp. 229-237.

[109] Q. Berthet et al., "An Area-Efficient SPHINCS+ Post-Quantum Signature Coprocessor," in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, USA, 2021, pp. 180-187.

[110] M. O. Saarinen, "Accelerating SLH-DSA by two orders of magnitude with a single hash unit," in *Proceeding Annual International Cryptology Conference*, Jan. 2024, pp. 276–304.

[111] S. Deshpande et al., "SPHINCSLET: An Area-Efficient Accelerator for the Full SPHINCS+ Digital Signature Algorithm," *ACM Transactions on Embedded Computing Systems* , Mar. 2025.

[112] T. Huang et al., "An Efficient and Reconfigurable Post-Quantum Crypto-Processor for SPHINCS+," *IEEE Transactions on Circuits and Systems I*, vol. 72, no. 5, pp. 2252-2262, May 2025.

[113] L. Beckwith, D. T. Nguyen, and K. Gaj, "High-Performance Hardware Implementation of Lattice-Based Digital Signatures," *Crypto. ePrint Arch.*, paper 2022/217, 2022.

[114] M. Schmid, D. Amiet, J. Wendler, P. Zbinden and T. Wei, "Falcon takes off - A hardware implementation of the Falcon signature scheme," *Cryptology ePrint Archive*, paper 2023/1885, Dec. 2023.

[115] Y. Ouyang et al., "FalconSign: An Efficient and High-Throughput Hardware Architecture for Falcon Signature Generation," *IACR Trans. on Cryptographic Hardware and Embedded Systems*, vol. 2025, no. 1, Dec. 2024.

[116] C. Aguilar-Melchor et al., "Towards Automating Cryptographic Hardware Implementations: A Case Study of HQC," in *Code-Based Cryptography Workshop*, Cham: Springer Nature Switzerland, 2023, pp. 62-76.

[117] C. Li, S. Song, J. Tian, Z. Wang and Ç. K. Koç, "An Efficient Hardware Design for Fast Implementation of HQC," in *2023 IEEE 36th International System-on-Chip Conference (SOCC)*, CA, USA, 2023, pp. 1-6.

[118] S. Deshpande et al. "Fast and efficient hardware implementation of HQC," in *International Conference on Selected Areas in Cryptography*, Cham: Springer Nature Switzerland, pp. 297-321, 2023.

[119] F. Antognazza, A. Barenghi, G. Pelosi and R. Susella, "A High Efficiency Hardware Design for the Post-Quantum KEM HQC," in *2024 IEEE Inter. Symp. on Hard. Oriented Security and Trust*, USA, 2024, pp. 431-441.

[120] F. Antognazza, A. Barenghi and G. Pelosi, "An Efficient and Unified RTL Accelerator Design for HQC-128, HQC-192, and HQC-256," *IEEE Transactions on Computer*, vol. 74, no. 7, pp. 2306-2320, Jul. 2025.

[121] S. Hong, "Side Channel Attacks," *MDPI*, Jun. 2019. doi: https://doi.org/10.3390/books978-3-03921-001-5.

[122] C. Su and Q. Zeng, "Survey of CPU Cache-Based Side-Channel Attacks: Systematic Analysis, Security Models, and Countermeasures," *Security and Communication Networks*, vol. 2021, Jun. 2021.

[123] P. C. Kocher, "Timing attacks on implementations of Diffie–Hellman, RSA, DSS, and other systems," in *Proc. Annual Int. Cryptol. Conf. Adv. Cryptol. (CRYPTO)*, pp. 104–113, Aug. 1996.

[124] B.-Y. Sim st al., "Chosen-ciphertext clustering attack on CRYSTALS-Kyber using the side-channel leakage of Barrett reduction," *IEEE Internet of Things Journal*, vol. 9, no.21, pp.21382-21397, Nov. 2022.

[125] H. Ma et al., "Vulnerable PQC against Side Channel Analysis-A Case Study on Kyber," in *2022 Asian Hardware Oriented Security and Trust Symposium*, Singapore, pp. 1-6, 2022.

[126] Y. Zhao et al., "Side Channel Security Oriented Evaluation and Protection on Hardware Implementations of Kyber," *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 70, no. 12, pp. 5025-5035, Dec. 2023.

[127] E. Karabulut, E. Alkim and A. Aysu, "Single-Trace Side-Channel Attacks on ω-Small Polynomial Sampling: With Applications to NTRU, NTRU Prime, and CRYSTALS-Dilithium," in *2021 IEEE Inter. Symposium on Hardware Oriented Secur. and Trust (HOST)*, pp. 35-45, 2021.

[128] H. Steffen et al., "Breaking and Protecting the Crystal: Side-Channel Analysis of Dilithium in Hardware," in *Inter. Conf. on Post-Quantum Cryptography 2023*, vol. 14154, pp. 688-711, Springer, Cham, 2023.

[129] Y. Wang et al., "Mind the Faulty KECCAK: A Practical Fault Injection Attack Scheme Applied to All Phases of ML-KEM and ML-DSA," *IEEE Trans. on Infor. Forensics and Security*, vol. 20, pp. 10035-10050, 2025.

[130] T. Schamberger et al., "A Power Side-Channel Attack on the Reed-Muller Reed-Solomon Version of the HQC Cryptosystem," *Cryptology ePrint Archive*, paper 2022/724, 2022.

[131] T. Huang et al., "A Timing Attack Resistant Lightweight Post-Quantum Crypto-Processor for SPHINCS+," in *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*, Singapore, 2024.

[132] P. Ravi et al., "Side-channel and fault-injection attacks over lattice-based post-quantum schemes (Kyber, Dilithium): Survey and new results," *ACM Trans. on Embedded Computing Systems*, vol. 23, issue 2, pp. 1-54, 2024.

[133] U. Banerjee, T. S. Ukyab, and A. P. Chandrakasan, "Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols," in *Proc. IACR Trans. Cryptograph. Hardw. Embedded Syst.*, 2019, pp. 17–61.

[134] G. Xin et al., "VPQC: A Domain-Specific Vector Processor for Post-Quantum Cryptography Based on RISC-V Architecture," *IEEE Trans. on Cirs. and Systems I: Reg. Papers*, vol. 67, no. 8, pp. 2672-2684, Aug. 2020

[135] Y. Zhu et al., "A 28nm 48KOPS 3.4μJ/Op Agile Crypto-Processor for Post-Quantum Cryptography on Multi-Mathematical Problems," in *2022 IEEE Inter. Solid-State Circuits Conf. (ISSCC)*, USA, 2022, pp. 514-516.

[136] A. Li et al., "A 40nm 2.76J/Op Energy-Efficient Secure Post-Quantum Crypto-Processor for Crystals-Kyber on ModuleLWE," in *IEEE Asian Solid-State Circuits Conference (A-SSCC)*, pp. 1-3, Nov. 2023.

[137] Y. Zhu et al., "A 28nm 69.4 kOPS 4.4 J/Op Versatile Post-Quantum Crypto-Processor Across Multiple Mathematical Problems," in *2024 IEEE Inter. Solid-State Circuits Conf. (ISSCC)*, USA, Feb. 2024, pp. 298-300.

[138] A. Li et al., "A 273W 0.34mm2 Efficient CRYSTALS-Kyber Processor for PQC Towards Edge Computing," in *2024 IEEE European Solid-State Electronics Research Conference (ESSERC)*, Belgium, 2024, pp. 472-475.

[139] Y. Zhao et al., "A high-performance domain specific processor with matrix extension of RISC-V for module-lwe applications," *IEEE Trans. on Cir. and Systems I: Regular Paper*, vol. 69, no. 7, pp. 2871–2884, 2022.

[140] P. Karl, J. Schupp, and G. Sigl, "Performance and communication cost of hardware accelerators for hashing in post-quantum cryptography," *ACM Trans. Embedded Comput. Syst.*, pp. 1–32, Jul. 2024.

[141] Z. Ye, X. Li, C. Wang, R. C. C. Cheung and K. Huang, "RVSLH: Acceleration of Postquantum Standard SLH-DSA With Customized RISC-V Processor," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 33, no. 7, pp. 1999-2003, July 2025.

[142] PQShield, available at: https://pqshield.com/products/pqp-cop/.

[143] E. kim (Jun. 2024), "SK Telecom Unveils Q-HSM, First Commercial Quantum Encryption Chip," *Business Korea*, available at: https://www.businesskorea.co.kr/news/articleView.html?idxno=219483.

[144] Samsung Electronics (Feb. 2025), "S3SSE2A: Hardware PQC Locks in Security for the Quantum Era", *Samsung Semiconductor Tech Blog*, available at: https://semiconductor.samsung.com/news-events/tech-blog/s3sse2a-hardware-pqc-locks-in-security-for-the-quantum-era/.

[145] N. Flaherty (Jul. 2025), "Three-in-one Post-Quantum Cryptography (PQC) Block Saves Area, Power," *eeNews Europe*, available at: https://www.eenewseurope.com/en/three-in-one-post-quantum-cryptography-pqc-block-saves-area-power/.

[146] SEALSQ, "SEALSQ's QS7001 secure chip to quantum-proof blockchain platforms," available at: https://www.sealsq.com/investors/news-releases/sealsqs-qs7001-secure-chip-to-quantum-proof-blockchain-platforms.

[147] NIST, "What Is Post-Quantum Cryptography?," *NIST* (updated Jun. 2025) [Online], available at: https://www.nist.gov/cybersecurity/what-post-quantum-cryptography.
[148] D. Butin, S.-L. Gazdag and J. Buchmann, "Real-world post-quantum digital signatures," in *Cyber Security and Privacy Forum*. Cham, Switzerland: Springer, 2015, pp. 41–52.
[149] H. Gharavi, J. Granjal and E. Monteiro, "Post-Quantum Blockchain Security for the Internet of Things: Survey and Research Directions," *IEEE Communication Surveys & Tutorials*, vol. 26, no. 3, pp. 1748–1774, 2024.
[150] Komodo Team, "Komodo integrates Dilithium: A quantum-secure digital signature scheme." Apr. 2022. [Online]. Available: https://komodoplatform.com/en/blog/dilithium-quantum-secure-blockchain/
[151] "Tidecoin post-quantum safe cryptocurrency–Tidecoin." Accessed: Jan. 1, 2024. [Online]. Available: https://tidecoin.co/

QUANG DANG TRUONG (Graduate Student Member, IEEE) received his B.S degree in Electrical and Electronic Engineering from Le Quy Don University of Technology, Vietnam, in 2019 and M.S degree in Electrical and Computer Engineering from Inha University, in 2023. He is currently pursuing his Ph.D. in Electrical and Computer Engineering from Inha University. His research interests include algorithm and architecture design for post-quantum cryptography, homomorphic encryption and digital signal processing.

HIEN NGUYEN (Graduate Student Member, IEEE) received her Bachelor's and M.S. degrees in Mathematics and Informatics Engineering from Hanoi University of Science and Technology, Vietnam, in 2020 and 2023. She is currently pursuing a Ph.D. degree in Informatics and Computing at the School of Informatics, Computing, and Cyber Systems, Northern Arizona University, USA. Her research interests include digital signatures, applied artificial intelligence, and post-quantum cryptography.

TUY TAN NGUYEN (Senior Member, IEEE) received the M.S. and Ph.D. degrees in Information and Communication Engineering from Inha University, South Korea, in 2016 and 2019, respectively. He is currently an Assistant Professor in the Department of Electrical and Computer Engineering, FAMU-FSU College of Engineering, Florida State University. From August 2022 to August 2025, he served as an Assistant Professor in the School of Informatics, Computing, and Cyber Systems, Northern Arizona University. Previously, he worked as a Senior Research Engineer at Conextt Inc., and as a Postdoctoral Fellow in the Department of Electrical and Computer Engineering, Inha University. Dr. Nguyen is a Technical Committee Member of the IEEE Circuits and Systems Society - Circuits and Systems for Communications. His research interests include post-quantum cryptography, homomorphic encryption, error correction codes, and applied artificial intelligence.

HANHO LEE (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees in electrical and computer engineering from the University of Minnesota, Minneapolis, USA, in 1996 and 2000, respectively. In 1999, he was a member of Technical Staff-1 with Lucent Technologies, Bell Labs, Holmdel, NJ, USA. From April 2000 to August 2002, he was a member of Technical Staff with Lucent Technologies (Bell Labs Innovations), Allentown, USA, where he was involved in the design of DSP multiprocessor architecture. From August 2002 to August 2004, he was an Assistant Professor with the Department of Electrical and Computer Engineering, University of Connecticut, USA. He has been a Faculty Member with Inha University, Incheon, South Korea, since September 2004, initially with the Department of Information and Communication Engineering and, since 2025, with the Department of Electrical and Electronic Engineering, where he is currently a Full Professor. He leads the Digital Integrated Systems Lab and is the Director of the Artificial Intelligence System on Chip Research Center, Inha University. He was a Visiting Researcher with the Electronics and Telecommunications Research Institute, South Korea, in 2005. He was a Visiting Scholar with Bell Labs, Alcatel-Lucent, Murray Hill, USA, from 2010 to 2011, and a Visiting Professor with The University of Texas at Dallas, USA, from 2017 to 2018. His research interests include algorithm and VLSI architecture design for post-quantum cryptography, homomorphic encryption, artificial intelligence, forward error correction coding, and digital signal processing. He served as the General Chair for ISICAS and the Technical Program Chair for ISCAS and APCCAS. He was the Chair of the IEEE Circuits and Systems for Communications Technical Committee. He was a Board of Governor of the IEEE Circuits and Systems Society (CASS) from 2020 to 2023. He is the Vice President of Technical Activities of the IEEE CASS.