A Full Threshold NIST PQC-Compliant Framework for Distributed Trust in Federal Public Key Infrastructure

Kiarash Sedghighadikolaei*§, Changqi Sun†§, Thang Hoang†, Bechir Hamdaoui‡, Attila A. Yavuz*
*University of South Florida, †Virginia Tech, ‡Oregon State University
*{kiarashs, attilaayavuz}@usf.edu, †{changqi, thanghoang}@vt.edu, ‡hamdaoui@eecs.oregonstate.edu

***Abstract*—The U.S. Federal Public Key Infrastructures (FPKI) relies on digital certificates and a network of Certificate Authorities (CAs) to build trust between federal agencies and their commercial partners. However, the current certificate generation process of FPKI faces two critical issues: The emerging quantum threats that jeopardize the security of conventional-secure digital signatures and the systemic vulnerabilities introduced by centralized signing operations. While threshold Post-Quantum (PQ) signatures offer a promising foundation for compromise-resilient trust, their adoption within the FPKI remains constrained by strict standard compliance requirements. ML-DSA is a prominent construction for thresholding within the NIST-PQ standards. However, existing threshold constructions of ML-DSA diverge from the NIST standard, limiting algorithmic compliance and hindering adoption in regulated settings such as FPKI, which require thorough cryptanalysis and revalidation. Hence, no fully standard-compliant threshold PQ solution currently meets FPKI's need for a distributed and quantum-resistant trust infrastructure.**

We introduce SHIELD, an efficient threshold NIST-PQ-compliant signature framework that replaces its classical centralized counterpart in certificate generation within the FPKI. SHIELD provides several key properties that overcome the limitations of prior approaches. First, it achieves algorithmic FIPS-204 compliance and seamless functional interchangeability with standard ML-DSA, without altering the signing algorithm. Second, it conforms to FPKI's structural and operational requirements, provides high security against adversarial environments, and prevents fraudulent certificates by mitigating single-key compromise risks. Finally, we provide the open-source implementation of SHIELD. We empirically evaluated the performance of SHIELD under diverse network latency conditions to validate its practical conformance with FPKI operational needs.

1. Introduction

The Federal Public Key Infrastructure (FPKI) [1] provides the U.S. Government with a common baseline for administering digital certificates and credentials that enable federated trust across government entities. Structurally, the FPKI forms a federated hierarchy comprising root (Common), intermediate, and issuing CAs that issue digital credentials (*e.g.,* Personal Identity Verification (PIV) certificates [2]) to federal agencies and approved commercial partners. Managed by the FPKI Policy Authority (FPKIPA) and operated by the FPKI Management Authority (FPKIMA), the FPKI ensures that all implemented mechanisms, including cryptographic components supporting federal authentication, digital identity, and inter-agency data exchange, comply with federal mandates (*e.g.,* FISMA [3], the E-Government Act [4]) and standards (*e.g.,* NIST SP 800-63-3 [5]) to keep operational continuity.

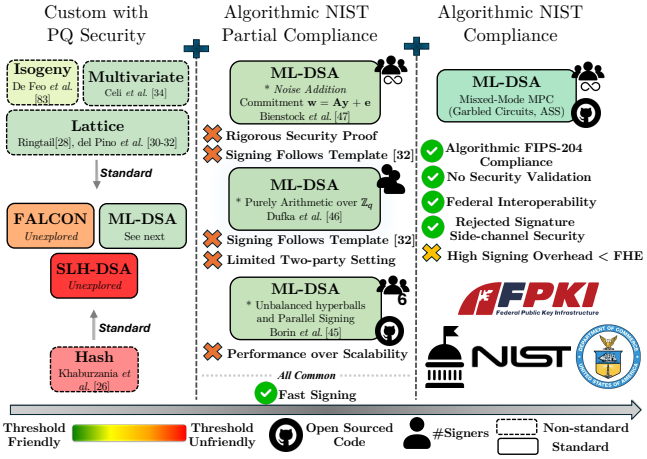

Figure 1: Roadmap of threshold PQ-secure signatures toward algorithmic NIST-compliance for FPKI.

Under the Federal Common Policy Framework [6], [7], CAs within the FPKI currently employ conventional digital signatures, such as RSA and ECDSA [8], for certificate issuance. However, reliance on these schemes exposes the FPKI to emerging quantum threats. Recent advances (*e.g.,* Google's Willow processor [9], the Quantum Echoes algorithm [10]) illustrate rapid progress toward efficient quantum computing. Such systems eventually may employ Shor's algorithm to break conventional cryptosystems [11].

To mitigate quantum threats, Post-Quantum (PQ) digital signatures have been developed [12], [13], [14], [15], [16], including the NIST Post Quantum Cryptography (NIST-PQC) standards ML-DSA [17], SLH-DSA [18], and FAL-

§*Equal contributions.*

CON [19] (with FIPS-206 forthcoming). However, their centralized deployment within the FPKI creates a systemic *single point of compromise* [20], [21], as each CA's private signing key, if compromised, could enable fraudulent certificate issuance and undermine trust across all federated domains. This has been evidenced by past PKI breaches [22], [23], [24] and the recent U.S. federal court system breach [25], which show that even well-defended infrastructures are vulnerable and threaten institutional trust.

Recognizing the centralization risk under quantum threats, numerous threshold post-quantum signatures have been proposed [26], [27], [28], [29], [30], [31], [32], [33], [34]. NIST has also announced the call for multi-party threshold schemes [35], which emphasizes that distributing trust across multiple parties is a critical requirement alongside a secure post-quantum transition. However, these threshold post-quantum signatures are not desirable for FPKI integration, as their underlying post-quantum signatures are not yet standardized. As the cornerstone of federal digital credentialing, the FPKI must adhere to federal standards and governance policies that define certificate issuance, cryptographic algorithms, and lifecycle management in accordance with established specifications [8], [17], [36], [37], [38], [39], [40], [41]. Any deviation from the exact algorithm would render the affected component noncompliant, undermining its security assurance, and thus require formal revalidation and costly cryptanalysis [42], [43].

Recent works have focused on NIST-standardized ML-DSA (FIPS-204) [44] to construct threshold post-quantum signatures [45], [46], [47] (see Figure 1 and Table 1). Despite their merits, they deviate from the ML-DSA's signing algorithm. For instance, some rely on the ML-DSA signing template [48] with message-independent nonce generation, while others simulate rejected transcripts to enable partial revelation (requiring further formal analysis). Although such deviations preserve verifiability under the ML-DSA verification algorithm [17] (*i.e.,* functional interchangeability), they remain *non-compliant* with the standard specification, and their adoption by the FPKI requires formal revalidation.

This leads us to a vital research question: *Can we design a compromise-resilient signature framework for the FPKI that simultaneously satisfies the following properties:* ***(1)*** *post-quantum security,* ***(2)*** *resilience to single-point of compromise,* ***(3)*** *functional interchangeability, and* ***(4)*** *algorithmic compliance with ML-DSA standard for FPKI operational requirements?*

1.1. Our Contributions

We answer the above research question affirmatively by designing SHIELD (*Secure NIST-compliant tHreshold post-quantum sIgnaturE for federaL PKI Digital-trust*), a NIST-compliant (algorithmically) threshold framework based on ML-DSA. SHIELD incorporates various secure multi-party computation (MPC) protocols [49], [50], [51] that execute each operation in its most suitable computing domain for optimal efficiency [52], [53]. In contrast to fully homomorphic encryption (FHE)-based approaches that enable single-round signing [54] but remain prohibitively expensive, our mixed-MPC approach achieves a more practical balance between computational and communication costs, making it a viable solution for real-world deployment. SHIELD achieves the following desirable properties:

• **Threshold Compliance with FIPS-204.** SHIELD, to the best of our knowledge, is the first *NIST-compliant (FIPS-204) threshold framework*, making it suitable for compromise-resilient distributed certificate signing in FPKI. The main ingredient that contrasts SHIELD from prior constructions lies in fully following the NIST standards algorithms. In other words, prior constructions offer only partial compliance (functional interchangeability). Moreover, their efforts to achieve efficient construction by avoiding distributed computation of complex functions (*e.g.,* sampling) led to deviations from standard protocols (*e.g.,* message-independent nonce precomputation) under various assumptions (*e.g.,* partial commitment revelation [47]). In contrast, FIPS-204 compliance requires faithfully and securely performing every operation mandated by the standard algorithm, whose security is already established without revalidation requirement, even if some of those operations are highly threshold-unfriendly.

To achieve this, SHIELD adopts a mixed MPC architecture: arithmetic-heavy routines (*e.g.,* polynomial arithmetic, including NTT-based multiplication) are executed using SPDZ [49] for high-throughput computation, while bit-heavy operations (*e.g.,* SHAKE256) leverage WRK17 [50]. This hybrid design achieves high efficiency without compromising NIST compliance, yielding a practical, fully standards-aligned threshold signing framework for the FPKI.

• **Performance Profiling and FPKI Deployment Feasibility.** We evaluate the performance of SHIELD, which enforces adherence to the NIST standard for distributed certificate signing, across a range of configurations involving multiple signing servers, diverse network conditions, and all NIST security levels using both fine-grained and system-level analysis. Our results show that SHIELD achieves constant-round signing across all security levels, with end-to-end latency under half a second among four servers, even in high-latency networks, and with low communication overhead. As we will discuss in detail in section 6, these performance characteristics fall well within the operational tolerances of FPKI workflows, demonstrating the practicality of SHIELD for federally regulated infrastructures.

• **Open-Source Implementation Artifacts.** We implemented SHIELD fully from scratch. Additionally, we implemented all bit-intensive components of ML-DSA in Verilog and synthesized them using Yosys [55] to obtain optimized gate-level netlists, thereby enhancing circuit efficiency. The Verilog codebase comprises ∼1400 lines, and the complete methodology, including source and synthesis files, testbenches, and a reference implementation, is available at:

https://github.com/open-threshold-pqc/SHIELD.git

• **Compromise Resilience Against Arbitrary Malicious Corruptions.** SHIELD provides *quantum-UC-secure* [56]

TABLE 1: Comparison of prior threshold ML-DSA vs. SHIELD's threshold ML-DSA for adoption in FPKI

Work	Algorithmic FIPS-204 Compliance	Free of Security Revalidation	Federal Interoperability	Base Work	Threshold Tools	Concealing Rejected Signature	Open Source Code (🐙)	Supported Parties (👥)
Borin *et al.* [45]	✗	✗	✗	Finally! [32]	Single (RSS)	✗	✓	6
Dufka *et al.* [46]	✗	✗	✗	Template [48]	Single (ASS)	✗	✗	2
Bienstock *et al.* [47]	✗	✗	✗	Template* [48]	Single (SSS)	✗	✗	∞
SHIELD (*Threshold ML-DSA*)	✓	✓	✓	FIPS-204 [17]	Mixed (ASS + GC)	✓	✓	∞

Abbreviations: RSS = Replicated Secret Sharing; ASS = Additive Secret Sharing; SSS = Shamir Secret Sharing; GC = Garbled Circuit.
Single: All signing occurs within one sharing domain. *Mixed (ASS + GC)*: combines Arithmetic (ASS) and Boolean (GC) MPC via efficient share conversion.
✓: Explicitly supported or demonstrated. ✗: unsupported or non-compliant. ∞: supports arbitrary parties; numeric values indicate reported maximum.
*Template**: Although the signing algorithm follows FIPS-204, the nonce is generated independently of the message, as in the template algorithm [48].

security against fully adversarial environments with an arbitrary number of servers controlled by the malicious adversary. This is ensured by the tolerance to dishonest majorities provided by SPDZ and WRK17. Unlike prior threshold constructions [46], [47], all ML-DSA's intermediate values in SHIELD remain in secure secret shares authenticated via information-theoretic message authentication codes (IT-MACs) [49], [57], eliminating potential leakages and providing resistance against side-channel attacks [58], [59], [60].

2. Models

In this section, we introduce SHIELD, an algorithmically FIPS-204-compliant threshold post-quantum digital signature framework.

2.1. System Model

As shown in Figure 2, SHIELD comprises n distributed server nodes (three illustrated) operated under a CA (*e.g.,* DoD Interoperability Root CA) within the FPKI. The SHIELD supports two main operations:

① **KeyGenDis** (*Key Generation and Distribution*): Given security parameter λ and the number of servers n, this operation generates a FIPS-204 key pair $(\mathsf{sk}, \mathsf{pk})$ and splits sk into n shares $\{[\![\mathsf{sk}]\!]_1, \ldots, [\![\mathsf{sk}]\!]_n\}$. In our system, we assume that CA is trusted during one-time setup and performs this initial key generation and secret sharing. The trusted CA then securely distributes the shares to the corresponding servers and subsequently deletes the original sk.

② **ThrSign** (*FIPS-204 Threshold Signing*): Given the to-be-signed message m (instantiated as TBSC [61] in FPKI), and all servers' private key shares $\{[\![\mathsf{sk}]\!]_1, \ldots, [\![\mathsf{sk}]\!]_n\}$, this operation, jointly executed by the servers, produces the FIPS-204-compliant signature serving the core component of the X.509 certificate in FPKI.

2.2. Threat Model

We consider a malicious, static, quantum-polynomial time (QPT) adversary $\mathcal{A}$ capable of corrupting up to $n-1$ signing servers, where malicious behavior includes injecting faulty information, selectively aborting phases (*e.g.,* during ThrSign), forking internal states to probe for secrecy leakage, and colluding among corrupted servers. The QPT aspect means that all computations of $\mathcal{A}$ run in time polynomial in the security parameter λ on a full quantum computer, capturing post-quantum threats that compromise classical digital signatures.

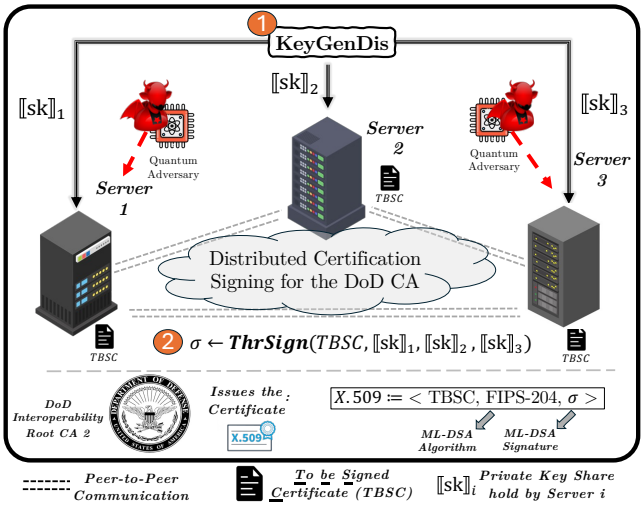

Figure 2: System overview of SHIELD. For simplicity, three servers are shown, though it generalizes to any numbers.

The adversary's goals are to break the confidentiality of the signing key or the unforgeability of the resulting signature. They can attempt to achieve these goals by interfering with the protocol's interactive phases: (i) Attacks before ThrSign: $\mathcal{A}$ may compromise one or more servers, extracting their respective shares of sk with the intent of reconstructing the entire private key and undermining the confidentiality guarantees of the key generation process. (ii) Attacks during ThrSign: This interactive phase is the primary target for $\mathcal{A}$. When executing the ThrSign, corrupted servers can deliberately send a malformed or computationally incorrect message to the non-compromised servers. The objective is

to trigger faulty responses that leak information about their respective private key shares. (iii) Attacks after ThrSign: This threat directly addresses the goal of forging a signature. After observing one or more legitimate executions of the protocol and collecting all associated public information and internal state from corrupted servers, $\mathcal{A}$ aims to forge a new, unapproved signature.

3. Building Blocks

In this section, we present the cryptographic building blocks underlying SHIELD, focusing on the core operations of the ML-DSA signature and MPC protocols.

Notation. Let $\mathbb{F}_p$ denote a prime field. For a prime $q \in \mathbb{Z}$, define the quotient ring $\mathbb{Z}_q := \mathbb{Z}/q\mathbb{Z}$. The polynomial ring $R_q = \mathbb{Z}_q[X]/\langle X^N + 1\rangle$ consists of polynomials of degree less than N with coefficients in $\mathbb{Z}_q$. We write $x \xleftarrow{\$} S$ for uniform sampling from a finite set S. For $a \in R_q$, the infinity norm is $\|a\|_\infty = \max_{i=1}^{N} |a_i|$, where a_i are the coefficients of a. For $\mathbf{b} \in R_q^k$, we define $\|\mathbf{b}\|_\infty = \max_{i=1}^{k} \|b_i\|_\infty$. A specific share of x is denoted by $[\![x]\!]^t$, where $t \in \{a, b\}$ indicates arithmetic ($\mathbb{Z}_p$) or Boolean ($\mathbb{Z}_2$) sharing, and general sharing of x is written as $[\![x]\!]$. Shares are generated as $\{[\![x]\!]_i^t\}_{i=1}^n \leftarrow \mathsf{Share}^t(x)$ and reconstructed as $x \leftarrow \mathsf{Reconstruct}^t(\{[\![x]\!]_i^t\}_{i=1}^n)$. Scalars are denoted by lowercase letters (*e.g.,* a), vectors by bold lowercase (*e.g.,* $\mathbf{a}$), matrices by bold uppercase (*e.g.,* $\mathbf{A}$), and we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$.

3.1. Overview of ML-DSA (FIPS 204)

The Module-Lattice-Based Digital Signature Standard (ML-DSA) [17] is a NIST-standardized, PQ-secure signature (FIPS-204 [17]) based on the hardness of MLWE and SelfTargetMSIS problems (see Appendix A). Derived from a Schnorr-like identification protocol via the Fiat-Shamir transform, it employs rejection sampling to ensure signatures are statistically independent of the secret key [62]. ML-DSA also leverages structured lattices for efficiency, enabling fast polynomial operations via the Number Theoretic Transform (NTT) and compact key sizes. We outline the ML-DSA:

1 $\mathsf{ML\text{-}DSA.KeyGen}(\xi)$: Given a 32-byte seed ξ, it deterministically derives a private-public key pair. The seed is expanded using an Extendable Output Function (XOF) ($\mathsf{SHAKE256}$) to obtain (ρ, ρ', K) for generating key components. The public matrix $\mathbf{A} \in R_q^{k \times \ell}$ is derived by $\mathsf{ExpandA}(\rho)$, and the secret vectors $\mathbf{s}_1 \in R_q^\ell$ and $\mathbf{s}_2 \in R_q^k$, with bounded coefficients (η), are sampled by $\mathsf{ExpandS}(\rho')$. The vector $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$ is computed and split into high-bits $\mathbf{t}_1$ and low-bits $\mathbf{t}_0$ via Power2Round. The private and public keys are $\mathsf{sk} = (K, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$ and $\mathsf{pk} = (\rho, \mathbf{t}_1)$, with public $\mathsf{tr} = \mathsf{H}(\mathsf{pk})$.

2 $\mathsf{ML\text{-}DSA.Sign}(\mathsf{sk}, M', \mathsf{rnd})$: Given the secret key sk, message M', and per-message randomness rnd, a seed $\rho'' \leftarrow \mathsf{H}(K \| \mathsf{rnd} \| \mu)$ is derived with $\mu \leftarrow \mathsf{H}(\mathsf{tr} \parallel M')$. A counter-based rejection sampling loop starts by deriving a nonce $\mathbf{y} \leftarrow \mathsf{ExpandMask}(\rho'')$. The commitment $\mathbf{w} = \mathbf{A}\mathbf{y}$ is computed, from which the high-order bits $\mathbf{w}_1$ are extracted. A hash $\tilde{c} = \mathsf{H}(\mu \| \mathsf{w1Encode}(\mathbf{w}_1))$ yields the challenge $c = \mathsf{SampleInBall}(\tilde{c})$. The response vector is $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$, and $\mathbf{r}_0 = \mathsf{LowBits}(\mathbf{w} - c\mathbf{s}_2)$. The loop is accepted if $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$ and $\|\mathbf{r}_0\|_\infty < \gamma_2 - \beta$. If accepted, hint $\mathbf{h} = \mathsf{MakeHint}(-c\mathbf{t}_0, \mathbf{w} - c\mathbf{s}_2 + c\mathbf{t}_0)$ is computed and signature $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$ is accepted if $\|c\mathbf{t}_0\|_\infty < \gamma_2$ and $\mathbf{h}$'s Hamming weight is $\leq \omega$.

3 $\mathsf{ML\text{-}DSA.Verify}(\mathsf{pk}, M', \sigma)$: Given the public key pk, message M', and signature σ, the verifier derives the challenge c from the commitment hash $\tilde{c}$ and uses it with $\mathbf{z}$ and the public key to compute an approximate commitment $\mathbf{w}'$. The hint $\mathbf{h}$ is used by UseHint to recover the high-order bits $\mathbf{w}_1'$, which are hashed with $\mu \leftarrow \mathsf{H}(\mathsf{tr} \parallel M')$ to derive the commitment hash $\tilde{c}'$. The signature is accepted if $\tilde{c}' = \tilde{c}$ and the response satisfies the bound $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$.

For details on ML-DSA's supporting components, see Appendix B. FIPS-204 defines security levels 2, 3, and 5 as ML-DSA-44, ML-DSA-65, and ML-DSA-87, respectively.

3.2. Mixed-Mode MPC

MPC [49], [50] allows multiple parties to compute a function over private inputs without revealing anything beyond the output. General-purpose MPC uses Boolean or arithmetic circuits, but mixed-mode MPC enhances efficiency by selecting the optimal representation for each operation (*e.g.,* Boolean for hashing, arithmetic for addition) with domain conversion via daBits [51]. We next describe the two protocols underlying our mixed-mode MPC:

SPDZ. The SPDZ protocol [49] is a preprocessing MPC protocol for arithmetic circuits providing active security with abort against a dishonest majority. In the offline phase, multiplication triples (Beaver triples) are generated, and in the online phase, the circuit is evaluated over private inputs. A secret $x \in \mathbb{F}_p$ is shared with its information-theoretic MAC (IT-MAC) as $[\![x]\!]_i^a = (x_i, \gamma_i(x))$, where $x = \sum_{i=1}^n x_i \bmod p$ and $\alpha x = \sum_{i=1}^n \gamma_i(x) \bmod p$, with α as the global MAC key. SPDZ supports local linear operations, $[\![c_1 x + c_2 y]\!]^a = c_1[\![x]\!]^a + c_2[\![y]\!]^a$ for constants $c_1, c_2 \in \mathbb{F}_p$, and share multiplication using Beaver triple: $[\![xy]\!]^a \leftarrow \mathsf{ShareMult}([\![x]\!]^a, [\![y]\!]^a)$ [49]. Additionally, SPDZ supports an efficient batch MAC checking mechanism [63], namely $\mathsf{BATCH\_MAC\_CHECK}(.)$, which verifies the integrity of the opened values (see section 4 for details). It takes the Agreed Random Values (denoted as $\mathsf{AGV}s$) and the locally stored partially opened values during protocol execution as inputs, and returns true if all partial openings pass the integrity test, and false otherwise.

WRK17. The WRK17 protocol [50] is a constant-round MPC for Boolean circuits, secure against up to $n-1$ malicious parties. It uses BDOZ-style authenticated sharing [57], where a bit x is shared as $[\![x]\!]_i^b$ with $x = \bigoplus_{i=1}^n x_i$. For a function f represented as a Bristol-format circuit [64], the protocol runs in two phases: an offline preprocessing phase, $\mathsf{WRK17.preprocess}(\mathsf{f})$, where parties collaboratively

construct an authenticated garbled circuit, and an online evaluation phase, $\mathsf{WRK17.online}(\mathsf{f}, \cdot)$, in which the circuit is securely evaluated over the shared inputs, producing authenticated shared output bits.

Cross-domain conversions. Domain conversion uses daBits $(\llbracket r \rrbracket^a, \llbracket r \rrbracket^b)$ [51], pairs of shares representing the same random bit in both domains. We denote the conversion of an authenticated share from arithmetic to Boolean using precomputed values $(\llbracket r_x \rrbracket^a, \llbracket r_x \rrbracket^b)$ as $\llbracket x \rrbracket^b \leftarrow \mathsf{A2B}(\llbracket x \rrbracket^a)$, and similarly, the conversion from Boolean to arithmetic with precomputed $(\llbracket r_y \rrbracket^a, \llbracket r_y \rrbracket^b)$ is written as $\llbracket y \rrbracket^a \leftarrow \mathsf{B2A}(\llbracket y \rrbracket^b)$.

4. Our Proposed SHIELD

In this section, we present the core operations of SHIELD: KeyGenDis and ThrSign, focusing on the latter, which enables algorithmic FIPS-204 compliance of the resulting signature. For ThrSign, we first give an overview, highlighting the use of mixed-mode MPC protocols and the strategic assignment of signing operations, as this assignment is the primary factor shaping overall performance. We then provide a detailed design analysis of it along with its performance costs, highlighting a cost-effective thresholding approach. Our design sets the threshold t equal to the total number of servers n, ensuring that all servers participate in signing, thereby maximizing trust and accountability.

4.1. Key Generation and Distribution

Since SHIELD is compliant with the FIPS-204 standard, the CA first generates an FIPS-204 key pair $(\mathsf{sk}, \mathsf{pk})$ and secret-share sk among the servers. In the KeyGenDis given the security parameter λ and the number of servers n, each server_i receives its share $\llbracket \mathsf{sk} \rrbracket_i$. The private key consists of a 32-byte key K and vectors $\mathbf{s}_1 \in R_q^\ell$, $\mathbf{s}_2 \in R_q^k$, and $\mathbf{t}_0 \in R_q^k$. The key K is Boolean-shared bit-wise, $\{\llbracket K \rrbracket_i^b\}_{i=1}^n \leftarrow \mathsf{Share}^b(K)$, while each polynomial vector is arithmetic-shared coefficient-wise over N coefficients. Since signing operations (*e.g.,* matrix multiplications) leverage the NTT for efficiency (to be detailed shortly), we first apply the NTT to $\mathbf{s}_1$, $\mathbf{s}_2$, and $\mathbf{t}_0$ before generating their arithmetic shares: $\{\llbracket \hat{\mathbf{s}}_1 \rrbracket_i^a\}_{i=1}^n \leftarrow \mathsf{Share}^a(\mathsf{NTT}_\mathsf{t}(\mathbf{s}_1))$, $\{\llbracket \hat{\mathbf{s}}_2 \rrbracket_i^a\}_{i=1}^n \leftarrow \mathsf{Share}^a(\mathsf{NTT}_\mathsf{t}(\mathbf{s}_2))$, $\{\llbracket \hat{\mathbf{t}}_0 \rrbracket_i^a\}_{i=1}^n \leftarrow \mathsf{Share}^a(\mathsf{NTT}_\mathsf{t}(\mathbf{t}_0))$. Finally, each share is securely distributed to its corresponding server via the CA.

Distributed Key Generation: While SHIELD assumes the key generation is done by a trusted CA, this process can also be thresholded using a distributed key generation based on mixed-mode MPC. Specifically, servers collaboratively compute the public seed ρ and shares of the secret seed ρ' and key K, and then derive the shares of secret $\mathbf{s}_1$ and $\mathbf{s}_2$ from ρ' (by evaluating hash functions under WRK17). Then servers can jointly compute the shares $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$ (via SPDZ), where $\mathbf{A}$ is derived from the public seed ρ, and then reconstruct the corresponding public key $\mathbf{t}$ from its shares.

4.2. Threshold Signing

Given the complexity of SHIELD's ThrSign, we begin with its overview. Then we provide an in-depth analysis of its design rationale, focusing on the threshold-friendliness of operations and their impact on performance. Before delving into ThrSign, we introduce the preprocessing required for the mixed-mode MPC protocol.

Before executing ThrSign, each server performs mixed-mode MPC preprocessing to generate the Correlated Random Values (CRVs) needed for the online phase (subsection 3.2), including daBits, Beaver triples, and AGVs for SPDZ batch MAC checking. In parallel, servers using the WRK17 preprocessing mechanism construct sufficient authenticated garbled circuits for all Boolean-heavy signing functionalities (*e.g.,* HighBits and SHAKE256).

4.2.1. ThrSign Overview. The ThrSign takes message m and private key share $\llbracket \mathsf{sk} \rrbracket_i$ from each server_i as the inputs and generates the signature. It enables distributed signing of signature by servers. Each server uses its $\llbracket \mathsf{sk} \rrbracket_i$ with m to produce a FIPS-204 signature share $\llbracket \sigma \rrbracket_i$. They send the shares to the CA that reconstructs the signature σ. The threshold protocol is summarized in Algorithm 1 (details omitted), with thresholded functionalities indicated by the subscript t. All intermediate values remain in arithmetic or Boolean shares, depending on their mapping to SPDZ or WRK17. For the public values such as public matrix $\mathbf{A}$, we just use plaintext representation rather than shared form.

Module operations, matrix-vector scalar-share multiplication (step 8), vector-share addition/subtraction (steps 12, 13, 18), and polynomial-vector share multiplication (steps 12, 13, 17, 18), are executed using the SPDZ protocol. These are implemented at the coefficient level: local SPDZ share addition and scalar-share multiplication, and interactive SPDZ share multiplication. In contrast, ML-DSA nonlinear functions, such as H, ExpandMask, and HighBits, are best implemented with the WRK17 protocol, operating on Boolean shares as input. As shown in the algorithm, we adopt a mixed-mode MPC approach that selects the most suitable protocol for each component, maximizing performance while preserving algorithmic FIPS-204 compliance without any deviation.

ML-DSA Template and FIPS-204 Compliance. As shown in Algorithm 1, the nonce $\mathbf{y}$ must be derived from a seed that depends on the message, which becomes available only during the online phase. Prior threshold ML-DSA works [45], [46], [47], [65] follow the original ML-DSA template [48], instead sampling the nonce independently of the message. This design enables nonce shares to be generated offline and avoids the costly MPC operations of H in step 5 and ExpandMask in step 7 that are seed-related. While efficient and valid, this method violates algorithmic FIPS-204 compliance and, hence, is unsuitable for FPKI adoption.

Intermediate Values and Side-Channel Attacks. All intermediate values remain in IT-secure arithmetic or Boolean

Algorithm 1: SHIELD.ThrSign:

$\sigma \leftarrow \mathsf{ThrSign}([\![\mathsf{sk}]\!]_i, M' = m)$

1: $([\![K]\!]^b,\ [\![\mathbf{s}_1]\!]^a, [\![\mathbf{s}_2]\!]^a, [\![\mathbf{t}_0]\!]^a) \leftarrow \text{unpack}[\![\mathsf{sk}]\!]_i$
2: $\mathbf{A} \leftarrow \mathsf{ExpandA}(\rho)$ ▷ ρ is public
3: $[\![\mu]\!]^b \leftarrow \mathsf{H_t}(M' \parallel \mathsf{tr})$ ▷ tr is public
4: $\mathsf{ctr} \leftarrow 0$ ▷ loop counter
5: $[\![\rho']\!]^b \leftarrow \mathsf{H_t}([\![K]\!]^b \parallel [\![\mu]\!]^b)$
6: **while** $([\![\mathbf{z}]\!]^a, [\![\mathbf{h}]\!]^b) = \perp$ **do** ▷ init $([\![\mathbf{z}]\!]^a, [\![\mathbf{h}]\!]^b) \leftarrow \perp$
7: $[\![\mathbf{y}]\!]^a \leftarrow \mathsf{ExpandMask_t}([\![\rho']\!]^b, i)$
8: $[\![\mathbf{w}]\!]^a \leftarrow \mathbf{A}\,[\![\mathbf{y}]\!]^a$
9: $[\![\mathbf{w}_1]\!]^b \leftarrow \mathsf{HighBits_t}([\![\mathbf{w}]\!]^b, 2\gamma_2)$
10: $[\![\tilde{c}]\!]^b \leftarrow \mathsf{H_t}([\![\mu]\!]^b \parallel [\![\mathbf{w}_1]\!]^b)$
11: $[\![c]\!]^a \leftarrow \mathsf{SampleInBall_t}([\![\tilde{c}]\!]^b)$
12: $[\![\mathbf{z}]\!]^a \leftarrow [\![\mathbf{y}]\!]^a + [\![c]\!]^a\ [\![\mathbf{s}_1]\!]^a$
13: $\mathbf{r}_0 \leftarrow \mathsf{LowBits_t}([\![[\![\mathbf{w}]\!]^a - [\![c]\!]^a\ [\![\mathbf{s}_2]\!]^a]\!]^b, 2\gamma_2)$
14: **if** $\|[\![\mathbf{z}]\!]^b\|_\infty \geq \gamma_1 - \beta$ or $\|[\![\mathbf{r}_0]\!]^b\|_\infty \geq \gamma_2$ **then**
15: $([\![\mathbf{z}]\!]^a, [\![\mathbf{h}]\!]^b) \leftarrow \perp$
16: **else**
17: $[\![\mathbf{h}]\!]^b \leftarrow \mathsf{MakeHint_t}\Big([\![-[\![c]\!]^a\ [\![\mathbf{t}_0]\!]^a]\!]^b,$
18: $[\![[\![\mathbf{w}]\!]^a - [\![c]\!]^a\ [\![\mathbf{s}_2]\!]^a + [\![c]\!]^a\ [\![\mathbf{t}_0]\!]^a]\!]^b,\ 2\gamma_2\Big)$
19: **if** $\|[\![c\mathbf{t}_0]\!]^b\|_\infty \geq \gamma_2$ or $\|[\![\mathbf{h}^{\#1s}]\!]^b\| > \omega$ **then**
20: $([\![\mathbf{z}]\!]^a, [\![\mathbf{h}]\!]^b) \leftarrow \perp$
21: **end if**
22: **end if**
23: $\mathsf{ctr} \leftarrow \mathsf{ctr} + \ell$
24: **end while**
25: Reconstruct $(\tilde{c}, \mathbf{z}, \mathbf{h})$ using $\mathsf{Reconstruct}^t(.)$ (see section 3)
26: $\sigma \leftarrow (\tilde{c}, \mathbf{z}, \mathbf{h})$
27: **return** σ

shares, with mixed-mode MPC and secure share conversion enabling evaluation without disclosure. This prevents exposure of values such as $\mathbf{w}_1$ or rejected signatures $(\tilde{c}, \mathbf{z}, \mathbf{h})$, mitigating recent side-channel attacks [58], [59], [60] that exploit such disclosures to recover the private key.

4.2.2. ThrSign Design Rationale. In this section, we provide an in-depth analysis of the ThrSign operation, breaking down each ML-DSA subroutine in the signing phase into core functionalities and showing how they are individually thresholded using SPDZ or WRK17. Figure 3 presents the complete threshold FIPS-204 Sign algorithm (step 2) with a breakdown of components: (i) thresholded components are marked with subscript t (*e.g.,* $\mathsf{H_t}$, $\mathsf{SampleInBall_t}$); (ii) SPDZ-thresholded components are shown in blue (*e.g.,* modular arithmetic); (iii) WRK17-thresholded components are shown in red (*e.g.,* HighBits); and (iv) share-conversion functions (A2B, B2A) appear in cyan. Each component is expanded in its box, with internal operations highlighted in a consistent color scheme. We next examine each operation in detail.

Hash Function and Streams 1. ML-DSA employs the SHAKE256 XOF [66] (denoted SHAKE hereafter) as its hash function. SHAKE follows the sponge construction: (i) init sets the 1600-bit state to zero; (ii) absorb reads input in RATE = 136-byte blocks, applying the Keccak-f permutation after each block to update state; (iii) finalize applies FIPS-202 padding; and (iv) squeeze iteratively applies Keccak-f, outputting RATE bytes per iteration until the desired length is reached. The init and finalize steps run locally since SHAKE is XOR-based [66], with Keccak-f as the sole non-linear step implemented via WRK17 3. Moreover, ML-DSA's stream functions for random byte sequences are thresholded using threshold SHAKE 2.

Cost: *SHAKE is XOR-based [66], with most operations computable locally under Boolean sharing; overhead arises only from thresholded* Keccak-f *calls via* WRK17*, scaling linearly with the number of calls. We note that these calls need not be performed independently and can be combined into a single functionality to reduce round complexity. For example, line 6 requires $k\ell$ calls, while line 11 requires 5-8 calls depending on the security level affecting* $\mathbf{w}_1$*'s rank.*

Nonce with ExpandMask 4. The nonce vector $\mathbf{y}$ is generated via threshold ExpandMask using seed $[\![\rho'']\!]^b$ and nonce ctr, producing ℓ polynomials. Each polynomial is obtained by generating $\lceil \frac{\mathsf{POLY\_SIZE}}{\mathsf{RATE}} \rceil$ RATE-byte blocks with SHAKE streams 2, then applying $\mathsf{bitUnpack}^*$ to extract coefficients in $[0, 2\gamma_1)$ to be in the field. POLY_SIZE is 576 for ML-DSA-44 and 640 for ML-DSA-65/87.

Cost: *Implemented via SHAKE streams, the overhead comes from* $\lceil \frac{\mathsf{POLY\_SIZE}}{\mathsf{RATE}} \rceil$ *threshold* Keccak-f *calls. In [46], [47], this step is done message-independently following the template of [48], removing its cost from the online phase and thereby losing full FIPS-204 compliance.*

Module Operations 5. In ML-DSA, the main module operations are (i) vector addition, adding polynomials in R_q component-wise, which reduces to coefficient-share additions, and (ii) matrix-vector or scalar-vector multiplication, carried out under the usual module rules. Matrix-vector multiplication appears in commitment generation (step 8) with the public matrix $\mathbf{A}$. Scalar-vector multiplication is used in response (steps 14-15) and in hint vector (step 22) computations, where the scalar c multiplies the vectors $\mathbf{s}_1$, $\mathbf{s}_2$, and $\mathbf{t}_0$ as polynomial arithmetic shares. With the NTT 6,7, these reduce to coefficient-wise share multiplications: ($\circ$) locally when one is constant (*e.g.,* $\mathbf{A}$), or ($\bullet$) via ShareMult when both are shares.

Cost: *Each* ShareMult *uses one Beaver triple, requiring ℓN triples in step 14 and kN in steps 15 and 22. We note that the calls to* ShareMult *can be performed in a single round to compute $c\mathbf{s}_1$, $c\mathbf{s}_2$, and $c\mathbf{t}_0$, without needing to wait for the computation of* MakeHint.

Decomposition to HighBits and LowBits 4. The commitment vector $\mathbf{w}$ is decomposed coefficient-wise into high and low bits using Decompose implemented via WRK17. The circuit for an individual decomposition takes a fixed 23-bit coefficient (23-bit q) and outputs the 23 low bits, with the number of high bits depending on the security level: (i) ML-DSA-44 outputs six high bits (29 bits total); (ii) ML-DSA-65/87 outputs four bits (27 bits total). Restricting outputs to the required high-bit size rather than 23 bits reduces both circuit depth and gate count.

Cost: *For $\mathbf{w} \in R_q^k$, the total computational overhead is kN calls to* Decompose*, done via* WRK17*. However,*

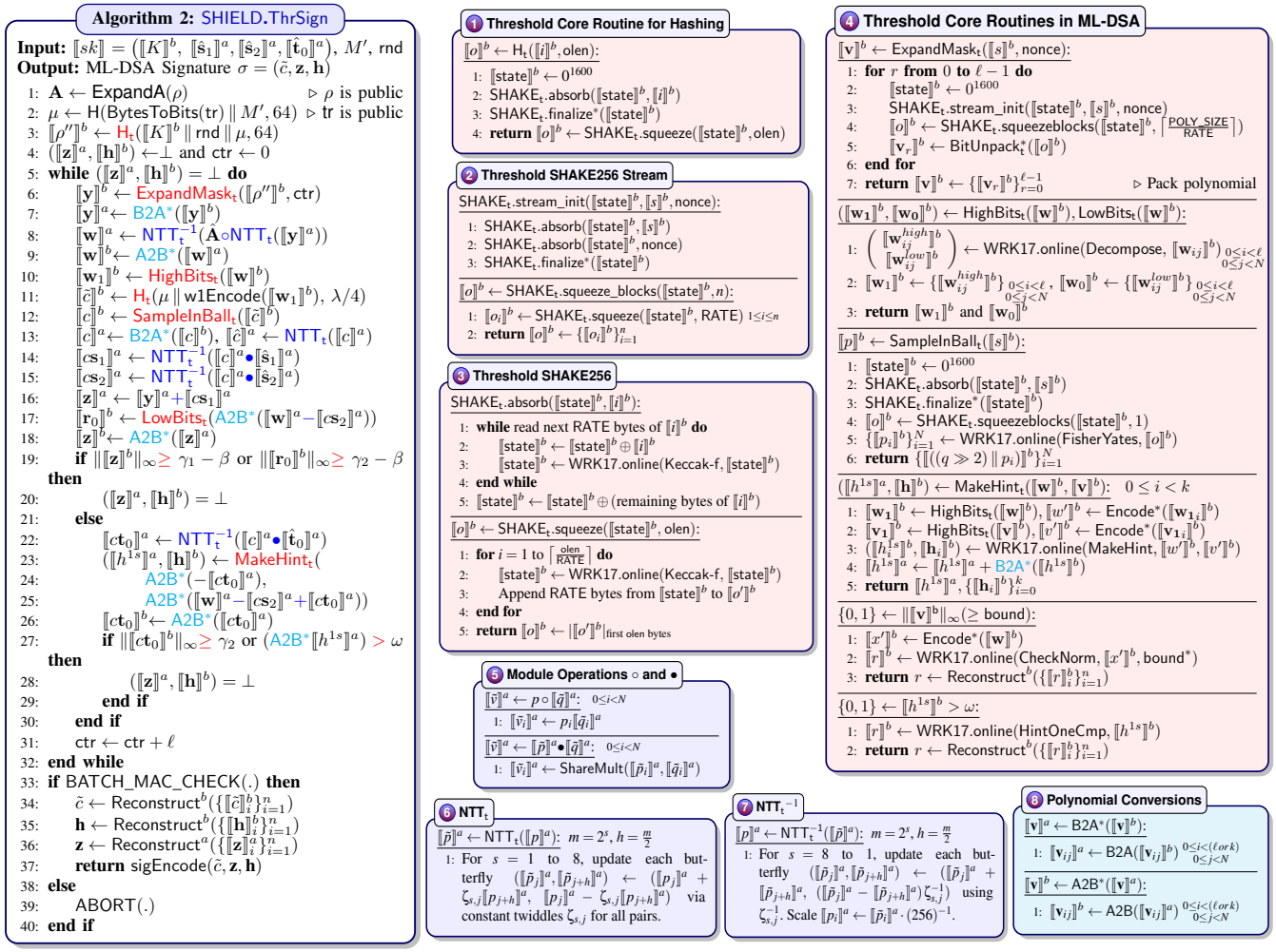

Figure 3: Design rationale of the SHIELD threshold signing implementation of algorithmically FIPS-204-compliant ML-DSA. Functions and parameters marked with * are highlighted for further discussion in the text.

given the independence of coefficients, this functionality is executed as a single comprehensive operation (all kN coefficients decomposed together) using WRK17 to reduce round complexity. Moreover, it should be combined with the subsequent Keccak calls at line 11 and with SampleInBall *(described next), since all of these are performed using WRK17 and require no conversion.*

Challenge Polynomial and FisherYates 4. The threshold SampleInBall pseudorandomly generates a polynomial share with at most τ nonzero coefficients (± 1) using the FisherYates shuffling algorithm. The FisherYates is implemented via WRK17 and consumes one block of random bytes for swapping index selection. These bytes are Boolean-shared and derived from SHAKE streams seeded with the commitment hash $[\![\tilde{c}]\!]^b$. The commitment hash (step 11) is produced by packing the commitment's coefficients with w1Encode, appending μ, and hashing with H, which is already thresholded via threshold SHAKE 3. We note that w1Encode packs by concatenating coefficient bytes, which applies equally to Boolean shared bytes. As in [17], one block of random bytes suffices with negligible failure probability, though an extra block of random indices may be added for safety.

An optimization in our FisherYates circuit avoids naively encoding each coefficient of the challenge polynomial with 23 bits. For $q = 8380417$, the coefficients $\{-1, 0, 1\}$ map to $(q-1)$, q, and $(q+1)$, differing only in the lower two bits; thus, each coefficient is encoded using two bits. After circuit evaluation, the full 23-bit values required for the arithmetic domain are reconstructed by having one signer append the upper 21 bits of q. In contrast, others append zeros, thereby preserving Boolean-share correctness. Experiments show that this optimization reduces the total gate count from approximately 890K to 355K for ML-DSA-44.

Cost: *Constructing the polynomial challenge requires one threshold SHAKE stream call to obtain a random byte block (one threshold* Keccak-f *invocation) and one threshold* FisherYates *call implemented via WRK17. We note that this*

operation is combined with the preceding HighBits *and* $\mathsf{H_t}$ *computations into a single functionality.*

Hint Vector and Hamming Weight ④. The hint vector is computed by threshold MakeHint, which takes two polynomial vector shares and, coefficient-wise, outputs whether adding the first to the second changes the latter's high bits. This can be done using WRK17 by packing all N Boolean-shared coefficients of both polynomials with Encode^* (adapted from w1Encode) and processing them in a single call. The first input, $[\![c\mathbf{t}_0]\!]^a$, has N 23-bits coefficients (5888 bits total); the second, $\mathbf{w}_1$, has N 6-bit coefficients (ML-DSA-44) or 4-bit (ML-DSA-65/87), yielding input sizes of 1536 and 1024 bits, respectively.

For all security levels, the circuit outputs 264 bits per polynomial in the hint vector $\in R_q^k$: the first $\log N = 8$ bits store a Boolean share of its Hamming weight, and the remaining 256 bits construct the polynomial share. The complete hint vector share is assembled in Boolean-shared format from all k polynomial shares. To compute the Hamming weight, parties convert the 8-bit shares from all k polynomials to arithmetic via A2B, sum locally to obtain a share of the weight, and convert back to Boolean shares with B2A. These are then passed to HintOneCmp implemented using WRK17 ④, which outputs zero if the weight is $\leq \omega$ and one otherwise.

Cost: *Computing the hint vector requires* k MakeHint *calls via* WRK17. *Checking the Hamming weight requires* kA2B + B2A *calls plus one* HintOneCmp *call. The addition circuit component of the* A2B *conversion is combined with* MakeHint *and* HintOneCmp *into a single functionality evaluated by WRK17. Furthermore, this can be integrated with the vector norm checking (described next) into one comprehensive circuit to further reduce communication rounds.*

Vector Norm Checking ④. Norm checking reduces to verifying each polynomial and computing the logical AND of the results. In threshold form, Boolean-shared coefficients are packed into 5888 bits using Encode^* and passed to CheckNorm evaluated via WRK17. The circuit outputs 1 for a bound violation and 0 otherwise, with the bound hardwired for simpler synthesis. In ML-DSA, norms are checked against γ_2, $\gamma_2 - \beta$, and $gamma_1 - beta$, requiring 9 circuits (3 per security level). After CheckNorm, parties reconstruct the result; if a violation is found, they output one, terminate further checks, and restart the loop.

Cost: *Norm checks over* $\mathbf{z}$, $\mathbf{h}$, *and* $c\mathbf{t}_0$ *require* ℓ CheckNorm *calls with bound* $\gamma_1 - \beta$, k *calls with bound* γ_2, *and* k *calls with bound* $\gamma_2 - \beta$, *respectively. We highlight that the vector norm checks for* $\mathbf{z}$, $\mathbf{h}$, *and* $c\mathbf{t}_0$ *can be combined into a single functionality and integrated into a larger circuit together with* MakeHint, *along with all required* A2B *addition circuits to achieve optimal round complexity.*

Polynomial Conversions ⑧. Vectors and scalars in the module are shared at the polynomial level and further at the coefficient level. Conversions between polynomials ($\mathsf{A2B}^*$ or $\mathsf{B2A}^*$) are performed per coefficient.

Cost: *Conversion overhead equals the number of* A2B *or* B2A *calls over all scalar or vector's coefficients. Each conversion uses a daBit, contributing to the total required daBits. We note that multiple coefficients are converted in batch rather than individually (fixed round). Furthermore, the addition circuit of* A2B *can be combined with the WRK17-mapped ML-DSA components into a single functionality (e.g., combining lines 9 and 10).*

4.2.3. Avoiding MAC-Key Leakage Vulnerability in Integrity Checking During Signing. Kyster *et al.* [67] identify a flaw in the SPDZ MAC-check protocol: in concurrent or multi-threaded executions, an adversary can extract the global MAC key α in one thread and reuse it to forge valid openings in another. While benign under single-threaded UC analysis, this attack compromises output integrity and, in some cases, input privacy. We eliminate this risk by applying the batch MAC-checking technique of [68] to all partial openings. After signing (Figure 3, step 32), each party P_i computes $\epsilon_i = \sum_{j=1}^{m} r_j [\![\gamma(s_j)]\!]_i - \left(\sum_{j=1}^{m} r_j s_j\right) [\![\alpha]\!]_i$ where m is the number of partial openings, r_j are AGVs, s_j the opened values, $[\![\gamma(s_j)]\!]_i$ the MAC shares, and $[\![\alpha]\!]_i$ the global MAC-key share. Each server broadcasts ϵ_i and verifies $\sum_{i=1}^{n} \epsilon_i = 0$, where n is the number of servers. If the check passes, all partial openings are valid. This single aggregated check replaces per-thread checks and closes the leakage vector reported in [67].

5. Implementation Details

We implemented SHIELD in C++. Its mixed-mode MPC protocol combines (i) a from-scratch implementation of SPDZ, (ii) WRK17, which we realized using the EMP-Toolkit [69], and (iii) from-scratch implementations of share conversion operations between the two protocols via daBits (A2B, B2A). The CA handles preprocessing and distributes key shares and CRVs under audit. For the underlying ML-DSA, we adopted the original reference implementation[1]. The key generation and signature verification use the unmodified algorithms, ensuring compatibility with signatures that SHIELD's ThrSign produces. For threshold signing, we preserved the original code structure and replaced each ML-DSA module (*e.g.,* CheckNorm) with its corresponding threshold-capable variant. To further verify the implementation, we used unit tests and NIST CMVP KAT files run under the reference implementation: all signatures generated by SHIELD were identical to the standard ones.

WRK17 Circuit Generation and Optimizations. To generate Bristol circuits for WRK17, we described each target functionality in Verilog (Combinational Logic) and compiled it to a gate-level netlist using Yosys [55], an open-source logic synthesis framework. This flow is generic and scalable, and Yosys optimizations significantly reduce circuit size and improve performance. For example, our synthesized 32-bit adder uses 294 gates and 358 wires, about 27% fewer gates than state-of-the-art implementations (EMP-Toolkit, OpenFHE, SCALE-MAMBA), which require 375 gates and

1. https://github.com/pq-crystals/dilithium

439 wires, while also leveraging more efficient XOR gates and fewer costly AND gates. We then converted the resulting netlist to Bristol format using a converter.

Building on this synthesis flow, we implemented all ML-DSA components thresholded using WRK17, resulting in approximately 1400 lines of Verilog code in total. We implemented each component as an independent Verilog module with configurable parameters, enabling modular testing. Among these components, the SampleInBall module proved to be the most complex to design, primarily due to its reliance on the FisherYates algorithm. In this implementation, all possible index values (256 possibilities for an 8-bit index) of the array for bit swapping must be considered in the circuit, since the circuit cannot terminate early. This requirement significantly increases the circuit's complexity and size, making it more challenging to synthesize. Appendix C summarizes the number of gates, input wires, and output wires for functionalities implemented using WRK17 based on our main analysis in section 4.

Following the same methodology, the reference ML-DSA implementation encodes polynomial coefficients as 32-bit integers, even though the modulus $q = 8380417$ fits within 23 bits. The extra bits introduce gates and wires in the Boolean circuits. In our Verilog implementation, we restricted coefficients to 23 bits, reducing circuit size and improving performance (*e.g.,* total gate reduction from 890K to 355K in SampleInBall for ML-DSA-44). Moreover, as we analyzed earlier (see section 4), we used shorter bit-lengths in certain functions (*e.g.,* HighBits, SampleInBall) to further shrink the circuits.

Batch Conversion and Multiplication. ML-DSA operates over modules where scalars are polynomials. In our mixed MPC framework, coefficients often require domain conversion, and polynomial multiplications (*e.g.,* $c\mathbf{s}_1$) invoke SMult each time if both coefficients are SPDZ shares. Naively, this incurs one round per coefficient, dominating runtime in high-latency settings. To address this, we implemented batch conversion and batch multiplication, collapsing many operations into a single round. For example, the A2B calls required before HighBits can be performed in a single round rather than once per coefficient of $\mathbf{w}$, reducing the cost from kN rounds to just one. As stated earlier in section 4, the remaining addition circuits are integrated with the ML-DSA operations as part of a comprehensive WRK17 functionality. Likewise, batch multiplication merges N coefficient multiplications of two polynomials into a single round, performing $c\mathbf{s}_1$ in one round instead of kN.

6. Experimental Evaluation

We evaluate SHIELD in two stages. First, we conduct a step-by-step analysis of the signing process, following the logical sequence of operations in a typical Fiat-Shamir-based digital signature. Second, we evaluate the overall system performance by measuring collaborative signature generation across multiple servers and under varying network conditions. Our analysis quantifies the round complexity, communication overhead, and computational cost.

6.1. Configuration

Hardware Setup. We benchmarked SHIELD with up to five servers, a configuration that is both sufficient and cost-effective for threshold operations [70]. All experiments were conducted on Ubuntu 24.04 systems, each equipped with an Intel Core i9-11900K CPU, 64 GB RAM, and a 1 TB SSD.

Network Setting. All experiments were performed within a local network using software-based emulation to reproduce different latency conditions: Low ($\approx$ 1 ms), Med ($\approx$ 3 ms), and High ($\approx$ 10 ms). We set network bandwidth at 10 Gbps.

Parameter Settings. SHIELD adopts the FIPS 204 parameter sets corresponding to all three NIST security levels for its threshold signing protocol, as summarized in Table 2.

TABLE 2: ML-DSA (FIPS-204) Parameter Configurations

Algorithm	(k, ℓ)	d	τ	γ_1	γ_2	η	β	ω
ML-DSA-44 *(NIST-2)*	(4, 4)	13	39	2^{17}	$\frac{q-1}{88}$	2	78	80
ML-DSA-65 *(NIST-3)*	(6, 5)	13	49	2^{19}	$\frac{q-1}{32}$	4	196	55
ML-DSA-87 *(NIST-5)*	(8, 7)	13	60	2^{19}	$\frac{q-1}{32}$	2	120	75

Modulus: $q = 8380417 = 2^{23} - 2^{13} + 1$. *Challenge Entropy*: $\log_2 \binom{256}{\tau} + \tau$ yields 192, 225, and 257 bits for NIST levels 2, 3, and 5, respectively. *Rejection Loops*: Expected iterations are 4.25, 5.1, and 3.81 for NIST levels 2, 3, and 5, respectively. Each polynomial consists of $N = 256$ coefficients, regardless of the security level.

6.2. SHIELD Fine-Grained Evaluation

In this section, we provide a step-by-step analysis of the SHIELD signing process, following the logical sequence of operations in a standard Fiat-Shamir-based digital signature. We further examine the generic MPC-based threshold components integrated into the protocol. For each operation, we characterize its round complexity, communication overhead, and computational cost in terms of CPU and memory usage. Recall that ML-DSA operates over a 23-bit modulus ($|q| = 23$) with a fixed dimension $N = 256$, while the parameters k and ℓ vary across security levels (see Table 2). We present the step-by-step analytical communication costs in Table 3 and the computational costs in Table 4 and Table 5. We note that a broadcast communication scheme is employed instead of a king-based model, reducing the number of communication rounds rather than keeping the communication overhead constant.

6.2.1. Fundamental MPC Operations. All WRK17-thresholded functionalities execute in a constant four-round online phase among n servers: three rounds for input processing and one for output. In Round 1, each server authenticates its input wire masks to all others using BDOZ MACs. In Round 2, each server broadcasts its masked input bit, and in Round 3, sends the corresponding κ-bit wire label to the evaluator, who performs the circuit evaluation locally, incurring a total communication of $2\kappa + n$ bits. In

WRK17, κ is set to 128-bits. During output processing, each party authenticates its output wire masks to the designated party via BDOZ MAC exchange. The resulting output wire values form Boolean shares, which can be directly used or converted to arithmetic shares via B2A. Computationally, XOR and INV gates are native Boolean operations, while AND gates, based on symmetric hashing, dominate the online evaluation time. In Appendix C we summarize the circuit complexity (type and number of gates) of WRK17-thresholded functionalities.

The MPC operation A2B consists of two steps: a partial opening of $|q|$ bits (one round), followed by a 23-bit addition circuit in WRK17 (see Appendix C for Mod23Add), which requires four rounds. In contrast, B2A and SMult complete in a single round, performing one partial opening of $|q|$ bits and two partial openings (total $2|q|$ bits), respectively. The conversion and multiplication are independent of the number of shares; that is, during batch conversion or multiplication, the round complexity remains same, while the communication cost scales linearly with the batch size.

TABLE 3: Analytical evaluation of online communication overhead across protocol operations.

Operation	Rounds	Comm. (bits)*
Derive ρ''	4	$1600I^{\mathsf{WRK}} + 512\kappa$
Sample $\mathbf{y}$	4	$516I^{\mathsf{WRK}} + 6528\ell\kappa$
Challenge c	4	$2kN\lvert q\rvert\, I^{\mathsf{WRK}} + 2N\kappa$
$c\mathbf{s}_1$, $c\mathbf{s}_2$, $c\mathbf{t}_0$	1	$2N(2k+\ell)\lvert q\rvert(n-1)$
CheckNorm and MakeHint**	4	$2(\ell+3k)N\lvert q\rvert\, I^{\mathsf{WRK}} + (2+kN)\kappa$

* Conversion-related communication costs are excluded from this table and discussed separately in the text. I^{WRK} represents the number of bits communicated per input bit in functionalities implemented using the WRK17 protocol, defined as $I^{\mathrm{WRK}} = 2\kappa + n - 1$, where $\kappa = 128$ bits is the size of the labels and BDOZ MACs employed in WRK17. ** CheckNorm and MakeHint include checking $\|\mathbf{z}\|_\infty$, $\|\mathbf{r}_0\|_\infty$, $\|c\mathbf{t}_0\|_\infty$, deriving $\mathbf{h}$, and comparing #1 against ω.

6.2.2. Core Threshold ML-DSA Operations. The core ML-DSA operations comprise sampling $\mathbf{y}$ using a private seed, performing module operations, computing the challenge, and executing norm checking and hint generation.

Module Operations. Once a polynomial is in NTT form, module operations in the threshold setting fall into two types: (i) SPDZ fast local scalar multiplications, and (ii) SPDZ share multiplications which require Beaver triples. After generating the challenge polynomial c, computing $c\mathbf{s}_1$, $c\mathbf{s}_2$, and $c\mathbf{t}_0$ requires $(2k+\ell)N$ triples, with each server broadcasting $2|q|$ bits per triple during multiplication.

Polynomial Conversion. The required conversions are as follows. The B2A conversions occur (i) for $\mathbf{y}$ when computing the commitment, requiring $\ell N|q|$ bits of communication; (ii) for c when computing $c\mathbf{s}_1$, $c\mathbf{s}_2$, and $c\mathbf{t}_0$, requiring $N|q|$ bits; and (iii) for $\mathbf{w}_0$ at line 17, requiring $kN|q|$ bits. Note that the B2A conversions for c and $\mathbf{w}_0$ can be completed in a single round. The A2B conversions are needed to process the commitment for HighBits ($\ell N|q|$ bits) and to convert $\mathbf{z}$, $\mathbf{r}_0$, $c\mathbf{t}_0$ for CheckNorm, and the first input of MakeHint, amounting to total $N(\ell+3k)|q|$ bits of communication. All A2B conversions for CheckNorm and MakeHint can be executed in batch. In the following, we show how these conversions can be further integrated with the core ML-DSA circuit to further reduce round complexity.

WRK17 Thresholded ML-DSA Operations. As mentioned before, all WRK17-based functionalities are completed in four rounds, with circuit evaluation executed locally by one server. The random seed ρ'' and vector $\mathbf{y}$ are computed independently; since ρ'' remains fixed across iterations, it requires no recomputation. Deriving ρ'' involves a call to the Keccak-f-1600 permutation, producing a 64-byte seed subsequently used to generate $\mathbf{y}$, where the circuit outputs $6\ell \times 1088$ bits (see section 4). Computing the challenge polynomial c entails generating and hashing the commitment, followed by the Fisher–Yates algorithm (Steps 8 and 11–12), all of which can be consolidated into a single WRK17 functionality with $kN|q|$ input bits. Similarly, verifying the norms of $|\mathbf{z}|_\infty$, $|\mathbf{r}_0|_\infty$, and $|c\mathbf{t}_0|_\infty$, along with constructing the hint vector $\mathbf{h}$ by counting ones and comparing against the threshold ω, can be integrated into one WRK17 functionality with $(\ell+3k)N|q|$ input bits.

We make two key observations. First, computing the challenge requires applying HighBits, which invokes A2B; likewise, all CheckNorm and MakeHint operations require A2B before evaluation, which in turn involves addition circuits. Since these additions occur within the garbled circuit, we integrate them into the core ML-DSA circuit to further reduce round complexity, resulting in a doubled communication cost ($2kN|q|$). Second, in successful iterations, all norm checks are fully evaluated. Hence, merging these operations balances computation and communication: early rejection reduces computation but increases rounds, while merging reduces round complexity but with higher computation.

In addition to the thresholded core ML-DSA operations and MPC-specific components, the final phases, BATCH_MAC_CHECK and all reconstructions performed via Reconstruct (Figure 3, steps 33-37), require two additional rounds in total [49], [50]. During signature reconstruction, the servers exchange signature shares equal in size to an ML-DSA signature, while the MAC batch check involves an exchange of $|q|$ bits.

In Table 4, we present the local evaluation time of the core ML-DSA components, including those implemented using WRK17, across NIST security levels 2-5. In our framework, only one server acts as the evaluator; hence, we report the CPU cycles and memory usage for this party. The derivation of the private seed ρ'' is independent of ML-DSA. Among all operations, deriving the expand mask is the most costly due to the multiple (6ℓ, see section 4 for constant 6) calls to the SHAKE.squeeze function, which internally invokes the Keccak-f permutation. Notably, deriving the challenge also incurs significant cost, as it requires coefficient decomposition along with hash calls to HighBits and SampleInBall. The decomposition for NIST-3 and NIST-5 is more lightweight than for NIST-2, while other operations

at these levels are more expensive. Consequently, the drop from 255 to 250 million cycles is notable, but this reduction is offset at NIST-5 by the increased number of hash calls and the higher cost of the FisherYates algorithm. In addition, in Table 5 we report the evaluator's memory usage during computation. For circuit-based operations, this includes storing the circuit representation in Bristol format (*i.e.,* gates and their input/output wires), the evaluation labels for all wires, the input labels sent by other signers to the evaluator, and the masked input bits. For non-circuit operations such as $c\mathbf{s}_1$, $c\mathbf{s}_2$, and $c\mathbf{t}_0$, memory usage stems from storing intermediate values required for partial openings.

TABLE 4: Signer's online CPU cycles (Million cycles) measured over one successful ML-DSA signing iteration.

Operation	Algorithm		
	ML-DSA-44	*ML-DSA-65*	*ML-DSA-87*
Derive ρ''	13	13	13
Sample $\mathbf{y}$	328	425	584
Challenge c	255	210	270
$c\mathbf{s}_1$, $c\mathbf{s}_2$, $c\mathbf{t}_0$	0.70	1.21	1.37
CheckNorm and MakeHint	110	162	216

A single Keccak-f permutation call takes 12.39 MCycles. In the above computations, we exclude the costs associated with conversions, including the addition circuits integrated into the ML-DSA components, to report the pure ML-DSA computation.

TABLE 5: Signer's online memory consumption (MB) measured over one successful ML-DSA signing iteration.

Operation	Algorithm		
	ML-DSA-44	*ML-DSA-65*	*ML-DSA-87*
Derive ρ''	$5.8 + 0.048n$	$5.8 + 0.048n$	$5.8 + 0.048n$
Sample $\mathbf{y}$	$159.1 + 0.41n$	$198.9 + 0.5n$	$278.4 + 0.7n$
Challenge c	$98.2 + 0.72n$	$88.6 + 1.08n$	$101.1 + 1.4n$
$c\mathbf{s}_1$, $c\mathbf{s}_2$, $c\mathbf{t}_0$	0.023	0.033	0.044
CheckNorm and MakeHint	$54.7 + 0.14n$	$78.4 + 0.2n$	$105.7 + 0.27n$

6.3. SHIELD System-Level Online Cost

We evaluate the *online performance* of SHIELD, where the servers collaboratively generate and reconstruct a joint signature. The *online signing latencies* for a single successful ML-DSA signing loop (see Table 2 for average loop counts) are measured across varying numbers of servers, *security levels*, and *network conditions*, as presented in Table 6. We distill the following *key takeaways*:

Algorithmic NIST Compliance. SHIELD provides NIST-compliant threshold ML-DSA secure against malicious adversaries by following FIPS-204 [17]. In contrast, prior works [45], [46], [47] trade algorithmic compliance for faster signing (*e.g.,* by using templates [48] or introducing threshold-friendly modifications), making them unsuitable for regulated FPKI.

Performance Within FPKI Operational Tolerance. The performance of SHIELD is consistent with the operational model of FPKI, where certificates are issued only during onboarding, re-keying, policy updates, or occasional renewals, rather than the high-frequency, ad-hoc setting of PKIs such as V2X [71]. These processes, governed by the FPKIPA, typically span days or weeks [72]; for example, the Department of State PKI requires at least two weeks notice for issuing a certificate and mandates artifact distribution within 24 hours [73]. These timelines highlight that certificate generation is a deliberate, long-lead process in which security takes precedence. Consequently, overall signing times of 256 ms or 418 ms under higher-latency networks for the 128-bit security level are feasible and consistent with the aforementioned operational model.

End-to-End Signing and Throughput. The reported values in Table 6 correspond to a single ML-DSA signing iteration. In practice, multiple repetitions are required to obtain a valid signature, constituting the end-to-end signing time, with average repetition counts of approximately 4.25, 5.1, and 3.85 for NIST-2, NIST-3, and NIST-5, respectively. Our evaluation hardware provides 16 cores, which can be leveraged to execute multiple signing attempts in parallel and return the first successful result, thereby improving throughput. For the 128-bit security level under low latency, we can run ≈ 4 parallel signing sessions, yielding an expected throughput of about 20 signatures per second; under high latency, this reduces to approximately 6 signatures per second. For higher security levels such as NIST-5 under low latency, we can run 4 parallel sessions, resulting in an expected throughput of about 12 signatures per second. Overall, throughput scales with available parallelism and repetition counts, while higher latency and stronger security primarily increase per-iteration time.

TABLE 6: Online signing latency (ms) of evaluator server during one successful ML-DSA signing iteration.

Algorithm	Latency Setting	# Servers			
		2	**3**	**4**	**5**
SHIELD *(NIST-2)*	Low	198	200	203	205
	Med	253	254	256	258
	High	414	416	418	418
SHIELD *(NIST-3)*	Low	246	247	250	252
	Med	294	295	296	298
	High	455	457	458	451
SHIELD *(NIST-5)*	Low	321	323	324	326
	Med	369	371	373	376
	High	530	531	533	536

In Table 7, we report the total number of rounds, communication overhead (as a function of n), and correlated random values. As shown in Table 3, all WRK17-based operations together require 17 rounds. Additional rounds include one for share multiplication to compute $c\mathbf{s}_1$, $c\mathbf{s}_2$,

TABLE 7: Online communication per server during signing for one successful signing iteration.

Algorithm	Rounds	Comm. (MB)	CRVs (count)		
			daBit	*Beaver Triple*	*Agreed Value*
SHIELD *(NIST-2)*	23	$7.7 + 0.045n$	7425	3072	11264
SHIELD *(NIST-3)*	23	$10.9 + 0.064n$	10496	4352	16128
SHIELD *(NIST-5)*	23	$14.7 + 0.087n$	14080	5888	21760

and $c\mathbf{t}_0$; one for B2A during commitment computation; and one for converting the challenge c and $\mathbf{w}_0$ for arithmetic operations. Furthermore, CheckNorm and MakeHint each require a partial opening during their A2B phase. Finally, one round is required for BATCH_MAC_CHECK and another for opening the signature shares, resulting in a total of 23 rounds. The analysis of correlated random values is provided in section 4.

6.3.1. Setup Costs: Key Generation and Preprocessing. Key generation in SHIELD follows the standard ML-DSA procedure and is independent of the number of servers. The CA constructs the key pair and distributes shares as discussed in section 4. The private key components $\mathbf{s_1}$, $\mathbf{s_2}$, and $\mathbf{t_0}$ are maintained in the NTT domain, and together with the key K, their arithmetic and Boolean shares are derived locally and distributed to all servers in a single round. Thresholded ML-DSA functions using WRK17 (see section 4) additionally require preprocessing to construct authenticated garbled circuits [50], which constitutes the primary costly phase of the framework. This step is not constant-round and scales with circuit size and the number of AND gates (see Table 9). However, unlike WebPKI or IoT PKIs, where issuers change frequently and operate continuously, FPKI involves infrequent issuance among a static set of servers, allowing preprocessing to be performed well in advance. We present our experimental analysis of this phase in the Appendix D.

6.4. Comparison with the State of the Art

We argue that our work is the first to achieve *algorithmic NIST compliance* in signing, thereby fulfilling the requirements of highly regulated environments such as the FPKI. Recent works [45], [46], [47] have achieved only *partial compliance* (functional interchangeability). A summary comparison is given in Table 8.

Our evaluation shows that partially compliant schemes achieve lower computational latency, ranging from 3-211 ms: [45] achieves 3-28 ms, [46] requires 155-211 ms in the 2-party setting, and [47] achieves 67-153 ms. In contrast, SHIELD requires 198-326 ms, reflecting the cost of evaluation in an algorithmic, NIST-compliant manner.

Algorithmic FIPS-204 compliance also increases communication. Our evaluation shows that SHIELD incurs 7.79-15.13 MB, compared to 0.09-0.57 MB for partially compliant schemes. This overhead stems from conversions between arithmetic and Boolean representations and the use of authentication tags to resist malicious adversaries during evaluation of the algorithm. In contrast, prior works operate entirely within optimized algebraic domains tailored to modified constructions. Memory usage follows a similar trend: [45] requires tens of KB to a few MB (*e.g.,* $\sim$2.2 MB for ML-DSA-65 in five-servers setting), [46] uses 55-96 MB for correlated randomness, and [47] incurs additional overhead for daBits [51], while SHIELD requires 160-280 MB due to authenticated circuit evaluation.

This gap extends to amortized costs as well. Specifically, [46], [47] have lightweight preprocessing of correlated randomness (*e.g.,* Beaver triples and daBits), while [45] reduces the online phase by one round through preprocessing. All of these approaches avoid the costly secure evaluation of hash functions by following the template-based constructions. Hence, the amortized cost is higher than that of prior works (see Table 10 in Appendix D). This overhead is due to following the FIPS-204 algorithm; however, it remains within the operational tolerances of FPKI certificate issuance.

Finally, we note that approaches leveraging FHE for threshold signing [54], [74] maintain strict adherence to the underlying signing algorithm, yet incur substantial computational overhead and produce very large ciphertexts (*e.g.,* $\sim$100 KB), rendering them impractical for FPKI.

TABLE 8: Online latency and communication complexity under the low-latency profile during one successful ML-DSA signing iteration.

Work	Rnds	#Servers							
		Computation (ms)				Communication (MB)			
		2	3	4	5	2	3	4	5
ML-DSA-44									
[45]	3	3.5	4	5	7	0.01	0.02	0.04	0.08
[46]	14	155	–	–	–	0.09	–	–	–
[47]	16	–	67	–	78	–	0.15	–	0.32
SHIELD	23	198	200	203	205	7.79	7.83	7.88	7.92
ML-DSA-65									
[45]	3	4	6	12	28	0.22	0.06	0.19	0.57
[46]	14	194	–	–	–	0.12	–	–	–
[47]	16	–	88	–	103	–	0.21	–	0.43
SHIELD	23	246	247	250	252	11.028	11.092	11.15	11.22
ML-DSA-87									
[45]	3	4.5	6	10	21	0.03	0.06	0.14	0.34
[46]	14	211	–	–	–	0.16	–	–	–
[47]	16	–	119	–	153	–	0.29	–	0.53
SHIELD	23	321	323	324	326	14.87	14.96	15.04	15.13

Rnds denotes the number of online communication rounds; — indicates unsupported settings (due to security assumptions or design constraints).

7. Security Analysis

We define the security of SHIELD using the quantum Universal Composability (quantum-UC) framework [56]. This ensures that an adversary $\mathcal{A}$ learns nothing beyond

the outputs of ideal functionality $\mathcal{F}$. The quantum-UC is a variant of the UC notion [75] and suitable for modeling quantum cryptography. The only difference between them is that it quantifies over quantum adversary, simulator, and environment instead of classical ones.

We define an ideal functionality $\mathcal{F}$ for SHIELD. Received the inputs $(\mathsf{ThrSign}, \mathsf{sk}, \mathsf{m})$, $\mathcal{F}$ outputs the signature σ. For the sake of simplicity we also omit other UC-related details in our functionality descriptions, like the use of sid.

We define the IDEAL and REAL worlds as following: At the beginning of the experiment for both worlds, the environment $\mathcal{Z}$ establishes the list of the corrupted server$_{i\in S}, S \subseteq [n]$. $\mathcal{Z}$ provides the inputs to the CA. $\mathcal{Z}$ sees the CA's outputs as soon as it receives the outputs from $\mathcal{F}$. $\mathcal{Z}$ gets the protocol transcript from the view of $\mathcal{A}$ or ideal simulator $\mathcal{S}$ at any time.

The Ideal World. In the IDEAL, the honest CA interacts only with $\mathcal{F}$, and $\mathcal{S}$ interacts with $\mathcal{F}$ on behalf of the corrupted servers. Upon inputs $\mathsf{ThrSign}, \mathsf{sk}$ and m from $\mathcal{Z}$, the CA forwards to $\mathcal{F}$. $\mathcal{F}$ queries $\mathcal{S}$ for $b_{\mathsf{tcg}} \in \{\mathsf{ok}, \perp\}$. If $b_{\mathsf{tcg}} = \mathsf{ok}$, $\mathcal{F}$ outputs the FIPS-204 signature to the CA; otherwise, it outputs $\perp$. We define $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\lambda)$ as the random bit representing the guess of $\mathcal{Z}$ on whether $\mathcal{S}$ view is from the ideal world execution or not.

The Real World. In the REAL, honest servers follow the real SHIELD protocol. Corrupted servers' actions are dictated by $\mathcal{A}$, which can cause aborts by deviating or withholding. Upon the inputs $\mathsf{ThrSign}, \mathsf{sk}$ and m from $\mathcal{Z}$, the CA executes the $\Pi.\mathsf{KeyGenDis}$ and all servers execute $\Pi.\mathsf{ThrSign}$. If the execution completes successfully, the CA outputs the signature σ to $\mathcal{Z}$; otherwise, it output $\perp$. We define $\mathrm{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda)$ as the random bit representing the guess of $\mathcal{Z}$ on whether $\mathcal{A}$ view is from the real world execution or not.

Definition 7.1 (Computational quantum-UC security of SHIELD)**.** The SHIELD scheme Π is computationally quantum-UC realizes the ideal functionality, $\mathcal{F}$ if, for any QPT real-world adversary $\mathcal{A}$, there exists a QPT ideal-world simulator $\mathcal{S}$ such that for every QPT environment $\mathcal{Z}$, there exists a negligible function negl with the security parameter λ such that:

$$|\Pr[\mathrm{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda) = 1] - \Pr[\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\lambda) = 1]| \leq \mathsf{negl}(\lambda).$$

To facilitate modular security analysis, we introduce a set of ideal sub-functionalities that our real protocol invokes. We denote by $\mathcal{F}_{\mathsf{AMPC}}$ an actively secure arithmetic MPC functionality supporting addition and multiplication over $\mathbb{F}_p$. It can be instantiated by SPDZ, typically in a preprocessing hybrid model for generating authenticated triples and SPDZ-style MACs. Similarly, $\mathcal{F}_{\mathsf{BMPC}}$ denotes an actively secure Boolean MPC functionality that supports bits operations. It can be instantiated by WRK17, typically in a function independent and dependent preprocessing hybrid model for generating the garbled tables and BDOZ-style MACs. Finally, $\mathcal{F}_{\mathsf{Conv}}$ represents an actively secure conversion functionality between arithmetic and Boolean shares. It can be instantiated by daBits, typically in a preprocessing hybrid model for generating random values in the arithmetic and Boolean domains.

Theorem 1 (Security of SHIELD)**.** *The* SHIELD Π *quantum-UC realizes* $\mathcal{F}$ *against static QPT adversary* $\mathcal{A}$ *corrupting up to* $n-1$ *servers.*

Proof. First, we show that $\Pi^{\mathcal{F}_{\mathsf{AMPC}},\mathcal{F}_{\mathsf{BMPC}},\mathcal{F}_{\mathsf{Conv}}}$ quantum-UC realizes $\mathcal{F}$. The hybrid protocol $\Pi^{\mathcal{F}_{\mathsf{AMPC}},\mathcal{F}_{\mathsf{BMPC}},\mathcal{F}_{\mathsf{Conv}}}$ is defined such that the components of SHIELD implemented by SPDZ are replaced with $\mathcal{F}_{\mathsf{AMPC}}$, those implemented by WRK17 are replaced with $\mathcal{F}_{\mathsf{BMPC}}$, and the share conversions implemented by daBits are replaced with $\mathcal{F}_{\mathsf{Conv}}$. We construct a simulator $\mathcal{S}$ to work on top of the $\mathcal{F}$, such that $\mathcal{Z}$ cannot distinguish whether it is playing with the REAL or IDEAL. Our simulator $\mathcal{S}$ runs $\mathcal{A}$ as a black box and realizes honest servers. $\mathcal{S}$ runs the $\Pi^{\mathcal{F}_{\mathsf{AMPC}},\mathcal{F}_{\mathsf{BMPC}},\mathcal{F}_{\mathsf{Conv}}}$ with the random key pair and message. Since the hybrid boxes reveal nothing but what ideal boxes allow, $\mathcal{Z}$'s view is indistinguishable. Second, the SPDZ UC-realizes $\mathcal{F}_{\mathsf{AMPC}}$ [49], the WRK 17 UC-realizes $\mathcal{F}_{\mathsf{BMPC}}$ [50], and daBits UC-realizes $\mathcal{F}_{\mathsf{Conv}}$ [51]. Since these sub-protocols provide information-theoretic security through the use of IT-MACs, their UC realizations also extend to the quantum setting. Therefore, by the universal composition theorem, we conclude that SHIELD Π quantum-UC realizes $\mathcal{F}$.

□

8. Related Work

Threshold signatures address the problem of single point of compromise (private key compromise) by distributing private keys among signers. Significant progress has been made in adapting traditional schemes for threshold signatures, including Schnorr [76], [77], [78] (enhancing throughput, adaptive security, and statelessness), ECDSA [79], [80] (efficient multiplicative-to-arithmetic conversions), and EdDSA [81], [82] (improving accountability, privacy, and proactive key refresh). With the growing need for post-quantum security, threshold signatures based on isogenies [83], hash functions [26], and multivariate assumptions, such as threshold variants of MAYO and UOV by Celi *et al*. [34], have emerged.

Building on this, threshold lattice-based signatures have enabled significant recent progress. At the core of efficient schemes such as the NIST-standardized ML-DSA lies the Fiat-Shamir with Abort transform. Unlike classical Schnorr signatures, its use in lattice-based settings requires responses to remain short to preserve security. This is achieved using: (i) Rejection Sampling [48], sampling from short vector distributions; (ii) Noise Flooding [33], adding fresh noise to conceal sensitive data; and (iii) Trapdoor Sampling [84], for compact signatures using Gaussian convolution.

Recent advances include Damgard *et al*. (two-round n-out-of-n protocol) [85] and Alkadri *et al*. [86], which reduces restarts without enlarging the parameters. In 2024, Chairattana-Apirom *et al*. [27] introduced a two-round threshold signature supporting arbitrary thresholds, improving communication efficiency at the cost of large secret key

shares. Tang *et al*. [29] presented the first t-out-of-n scheme with proactive key refresh, leveraging a variant of SPDZ and distributed rejection sampling. Speaking of scalability, Del Pino *et al*. [87] proposed TRACCOON, a three-round protocol supporting up to 1024 signers with pairwise one-time additive masks for key-leakage prevention. Katsumata [88] achieved adaptive security in five rounds under MLWE and MSIS assumptions. Eliminating the need for non-standard assumptions, Boschini *et al*. [28] proposed a two-round scheme, while Zhu *et al*. [89] introduced the AOM-MISIS assumption to enable tighter proofs for two-round lattice-based threshold signatures. Subsequent optimizations with adding lightweight Identifiable Abort (IA) by Del Pino *et al*. [30], [31], [32] yielded TRACCOON-IA and a framework removing NIZKs, leading to the most efficient lattice-based threshold signature with IA.

Among the NIST-standardized post-quantum signatures, ML-DSA [17] stands out as the most suitable candidate for threshold deployment within PKI environments. SLH-DSA [18] incurs large signature sizes (up to 17 KB) and slow signing performance due to its heavy reliance on hash-based operations, while FALCON [19] (with FIPS-206 forthcoming) depends on floating-point arithmetic, complicating secure threshold implementations. The recent integration of ML-DSA within the X.509 certificate ecosystem [90] further reinforces its practicality for certificate infrastructures. Recent works on threshold ML-DSA [45], [46], [47] achieve only partial algorithmic compliance, limited to functional interchangeability, as their signing deviates from FIPS-204. Consequently, these schemes cannot be deployed in regulated FPKI domain.

9. Conclusion

In this work, we proposed SHIELD, an algorithmically NIST-compliant threshold post-quantum signature based on standardized ML-DSA, to enhance the compromise resilience of FPKI. Unlike prior approaches that sacrifice compliance, SHIELD conforms entirely to FIPS-204, producing signatures functionally identical to ML-DSA without modifying the signing algorithm. The framework achieves post-quantum security, resilience to single-point compromise, functional interchangeability, and FIPS-204 compliance. Comprehensive evaluation across all NIST security levels and diverse network conditions demonstrates the practical efficiency of SHIELD for FPKI. By uniting standard compliance, malicious security, and practical deployability, SHIELD provides a concrete and verifiable path for transitioning the FPKI to a quantum-resistant trust infrastructure with distributed, compromise-resilient signing. Our open-source implementation enables adoption and verification by the community and regulated environments.

Acknowledgement

This work is supported by the National Science Foundation under grants CNS-2350213, CNS-2350214, and CNS-2350215, as well as by the Cisco Research Award (290003).

Ethics considerations

None

References

[1] Federal Public Key Infrastructure Policy Authority, "Federal public key infrastructure (fpki) trust infrastructure overview," https://s3.amazonaws.com/sitesusa/wp-content/uploads/sites/1171/2017/01/FPKI-Trust-InfrastructureOverview.pdf, December 2016, accessed: 2025-08-02.

[2] H. Ferraiolo, A. Regenscheid, and J. Fenton, "Guidelines for derived personal identity verification (piv) credentials," National Institute of Standards and Technology, Tech. Rep., 2024.

[3] United States Congress, "Federal information security modernization act of 2014," *Public Law 113–283, 128 Stat. 3073*, 2014, enacted December 18, 2014. Codified at 44 U.S.C. § 3551 *et seq.* [Online]. Available: https://www.congress.gov/113/plaws/publ283/PLAW-113publ283.pdf

[4] "E-government act of 2002," 116 Stat. 2899, United States Public Law 107–347, Dec. 2002. [Online]. Available: https://www.congress.gov/107/plaws/publ347/PLAW-107publ347.pdf

[5] P. A. Grassi, M. E. Garcia, and J. L. Fenton, "Draft nist special publication 800-63-3 digital identity guidelines," *National Institute of Standards and Technology, Los Altos, CA*, 2017.

[6] Federal PKI Policy Authority, "X.509 certificate policy for the u.s. federal pki common policy framework," PDF, IDManagement.gov / Federal PKI Policy Authority, Tech. Rep., 2020, version 2.3, published October 25, 2024 – accessible online. [Online]. Available: https://www.idmanagement.gov/docs/fpki-x509-cert-policy-common.pdf

[7] "X.509 certificate policy for the federal bridge certification authority," Federal Public Key Infrastructure Policy Authority (FPKIPA), Washington, D.C., Technical Report Version 3.8, Aug. 2025, clarifies KRA and KRO authentication requirements, and updates the definition of PKI Sponsor. Changes effective immediately. [Online]. Available: https://www.idmanagement.gov/docs/fpki-x509-cert-policy-fbca.pdf

[8] National Institute of Standards and Technology, "FIPS 186-5: Digital Signature Standard (DSS)," U.S. Department of Commerce, National Institute of Standards and Technology, Gaithersburg, MD, Federal Information Processing Standards Publication 186-5, Feb. 2023, effective February 3, 2023; supersedes FIPS 186-4. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5.pdf

[9] H. Neven. (2024, December) Meet willow, our state-of-the-art quantum chip. Google Quantum AI. Google Research Blog, accessed October 24, 2025. [Online]. Available: https://blog.google/technology/research/google-willow-quantum-chip/

[10] "Observation of constructive interference at the edge of quantum ergodicity," *Nature*, vol. 646, no. 8086, pp. 825–830, 2025.

[11] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th annual symposium on foundations of computer science*. Ieee, 1994, pp. 124–134.

[12] J. W. Bos, O. Bronchain, L. Ducas, S. Fehr, Y.-H. Huang, T. Pornin, E. W. Postlethwaite, T. Prest, L. N. Pulles, and W. van Woerden, "Hawk specification document, version 1.0," NIST PQC Digital Signature Project, Technical Report Round 1/spec-files/hawk-spec-web.pdf, June 2023, version 1.0. [Online]. Available: https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/hawk-spec-web.pdf

[13] M. Baldi, A. Barenghi, M. Battagliola, S. Bitzer, M. Gianvecchio, P. Karl, F. Manganiello, A. Pavoni, G. Pelosi, F. Pintore, P. Santini, J. Schupp, E. Signorini, F. Slaughter, A. Wachter-Zeh, and V. Weger, "Codes and restricted objects signature scheme: Submission to the nist post-quantum cryptography standardization process," CROSS Team, Ancona, Italy, Technical Report, July 2025, submitted to the NIST Post-Quantum Cryptography Standardization Process. [Online]. Available: https://www.cross-crypto.com/CROSS_Specification_v2.2.pdf

[14] M. Baldi, A. Barenghi, L. Beckwith, J.-F. Biasse, T. Chou, A. Esser, K. Gaj, P. Karl, K. Mohajerani, G. Pelosi, E. Persichetti, M.-J. O. Saarinen, P. Santini, R. Wallace, and F. Zweydinger, "Linear equivalence signature scheme (less)," LESS Project, Florida Atlantic University, Florida Atlantic University, Boca Raton, USA, Tech. Rep., 2025, submitted to NIST Post-Quantum Cryptography Standardization. URL: https://www.less-project.com/LESS-2025-02-07.pdf.

[15] M. A. Aardal, G. Adj, D. F. Aranha, A. Basso, I. A. Canales Martínez, J. Chávez-Saab, M. Corte-Real Santos, P. Dartois, L. De Feo, M. Duparc, J. K. Eriksen, T. B. Fouotsa, D. L. Gazzoni Filho, B. Hess, D. Kohel, A. Leroux, P. Longa, L. Maino, M. Meyer, K. Nakagawa, H. Onuki, L. Panny, S. Patranabis, C. Petit, G. Pope, K. Reijnders, D. Robert, F. Rodríguez-Henríquez, S. Schaeffler, and B. Wesolowski, "SQIsign," National Institute of Standards and Technology, Tech. Rep., 2025. [Online]. Available: https://sqisign.org

[16] G. Adj, N. Aragon, S. Barbero, M. Bardet, E. Bellini, L. Bidoux, J.-J. Chi-Domínguez, V. Dyseryn, A. Esser, T. Feneuil, P. Gaborit, R. Neveu, M. Rivain, L. Rivera-Zamarripa, C. Sanna, J.-P. Tillich, J. Verbel, and F. Zweydinger, "Mirath signature scheme," Technology Innovation Institute, Naquidis Center, Politecnico di Torino, LITIS, INRIA, Télécom Paris, Cryptoexperts, University of Limoges, Tech. Rep., September 2025, submitters listed in alphabetical order. Contact: team@pqc-mirath.org. [Online]. Available: https://pqc-mirath.org/assets/downloads/mirath_v2.1.0.pdf

[17] N. I. of Standards, T. (NIST), T. Dang, J. Lichtinger, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, R. Perlner, and A. Robinson, "Module-lattice-based digital signature standard," 2024.

[18] N. I. of Standards, T. (NIST), and D. Cooper, "Stateless hash-based digital signature standard," 2024.

[19] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, Z. Zhang *et al.*, "Falcon: Fast-fourier lattice-based compact signatures over ntru," *Submission to the NIST's post-quantum cryptography standardization process*, vol. 36, no. 5, 2018.

[20] F. P. P. Authority, "Federal public key infrastructure (fpki) incident management plan," Federal PKI Policy Authority, Washington, DC, USA, Tech. Rep. Version 3.0, Sep. 2020, published September 4, 2020. [Online]. Available: https://www.idmanagement.gov/docs/fpki-imp.pdf

[21] Microsoft, "Removal of the U.S. federal common policy CA certificate from the microsoft trusted root," https://learn.microsoft.com/en-us/troubleshoot/windows-server/certificates-and-public-key-infrastructure-pki/microsoft-trusted-root-store-removal-of-us-federal-common-policy, Microsoft Learn, January 2025, accessed: 2025-10-25.

[22] J. Nightingale, "Comodo Certificate Issue – Follow Up – Mozilla Security Blog," https://shorturl.at/S1LxF, 2011, [Accessed 2025-07-19].

[23] D. Fisher, "Final Report on DigiNotar Hack Shows Total Compromise of CA Servers," https://shorturl.at/hcmKb, 2012, [Accessed 19-07-2025].

[24] D. Biswas, "What Happens When A Certificate Chain of Trust Breaks?" https://shorturl.at/bWcKI, 2022, [Accessed 2025-07-19].

[25] Reuters, "U.s. federal court filing system breached in sweeping hack, politico reports," *Reuters*, Aug. 2025, accessed: 2025-11-09. [Online]. Available: https://tinyurl.com/3dju7j5k

[26] I. Khaburzaniya, K. Chalkias, K. Lewi, and H. Malvai, "Aggregating and thresholdizing hash-based signatures using starks," in *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, 2022, pp. 393–407.

[27] R. Chairattana-Apirom, S. Tessaro, and C. Zhu, "Partially non-interactive two-round lattice-based threshold signatures," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2024, pp. 268–302.

[28] C. Boschini, D. Kaviani, R. W. Lai, G. Malavolta, A. Takahashi, and M. Tibouchi, "Ringtail: practical two-round threshold signatures from learning with errors," in *IEEE Symposium on Security and Privacy (SP)*. IEEE, 2025, pp. 149–164.

[29] G. Tang, B. Pang, L. Chen, and Z. Zhang, "Efficient lattice-based threshold signatures with functional interchangeability," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 4173–4187, 2023.

[30] R. del Pino, T. Espitau, G. Niot, and T. Prest, "Simple and efficient lattice threshold signatures with identifiable aborts," *Cryptology ePrint Archive*, 2025.

[31] R. del Pino, S. Katsumata, G. Niot, M. Reichle, and K. Takemure, "Unmasking traccoon: A lattice-based threshold signature with an efficient identifiable abort protocol," *Cryptology ePrint Archive*, 2025.

[32] R. Del Pino and G. Niot, "Finally! a compact lattice-based threshold signature," in *IACR International Conference on Public-Key Cryptography*. Springer, 2025, pp. 169–199.

[33] R. del Pino, S. Katsumata, M. Maller, F. Mouhartem, T. Prest, and M. Saarinen, "Threshold raccoon: Practical threshold signatures from standard lattice assumptions (2024), to appear in eurocrypt 2024."

[34] S. Celi, D. Escudero, and G. Niot, "Share the mayo: thresholdizing mayo," in *International Conference on Post-Quantum Cryptography*. Springer, 2025, pp. 165–198.

[35] L. T. A. N. Brandão and R. Peralta, "Nist first call for multi-party threshold schemes," National Institute of Standards and Technology, Gaithersburg, MD, NIST Internal Report NISTIR 8214C 2pd, Mar. 2025, second Public Draft. [Online]. Available: https://doi.org/10.6028/NIST.IR.8214C.2pd

[36] U.S. General Services Administration, "Federal pki governance and compliance audit information," https://www.idmanagement.gov/fpki/, 2025, accessed: 2025-10-28. [Online]. Available: https://www.idmanagement.gov/fpki/

[37] U.S. General Services Administration, Office of Government-wide Policy, "FICAM Program," https://www.idmanagement.gov/ficam/, 2025, accessed: 2025-10-28.

[38] E. Barker and A. Roginsky, "Transitioning the use of cryptographic algorithms and key lengths," National Institute of Standards and Technology, Gaithersburg, MD, NIST Special Publication 800-131Ar3 ipd, October 2024, initial Public Draft. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-131Ar3.ipd

[39] E. Barker, "Recommendation for key management part 1 general," National Institute of Standards and Technology, Gaithersburg, MD, NIST Special Publication SP 80057 Part 1 Rev 5, May 2020, uS Department of Commerce Computer Security Division Information Technology Laboratory. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-57pt1r5

[40] G. M. Raimondo and L. E. Locascio, "FIPS 203 federal information processing standards publication module-lattice-based key-encapsulation mechanism standard," *NIST Natl. Inst. Stand. Technol*, vol. 47, 2024.

[41] N. I. of Standards and Technology, "Security requirements for cryptographic modules (fips pub 140-3)," US Department of Commerce, National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep. FIPS PUB 140-3, 2019, supersedes FIPS PUB 140-2. [Online]. Available: https://doi.org/10.6028/NIST.FIPS.140-3

[42] L. N. Pulles and M. Tibouchi, "Cryptanalysis of eaglesign," in *International Conference on Security and Cryptography for Networks*. Springer, 2024, pp. 165–186.

[43] J. Devevey, O. Fawzi, A. Passelègue, and D. Stehlé, "On rejection sampling in lyubashevsky's signature scheme," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2022, pp. 34–64.

[44] B. Westerbaan and L. Valenta. (2024) A look at the latest post-quantum signature standardization candidates. Cloudflare. Accessed: 2025-11-07. [Online]. Available: https://blog.cloudflare.com/another-look-at-pq-signatures/

[45] G. Borin, S. Celi, R. del Pino, T. Espitau, G. Niot, and T. Prest, "Threshold signatures reloaded: Ml-dsa and enhanced raccoon with identifiable aborts," *Cryptology ePrint Archive*, 2025.

[46] A. Dufka, S. Kravtšenko, P. Laud, and N. Snetkov, "Trilithium: Efficient and universally composable distributed ml-dsa signing," *Cryptology ePrint Archive*, 2025.

[47] A. Bienstock, L. de Castro, D. Escudero, A. Polychroniadou, and A. Takahashi, "Efficient, scalable threshold ml-dsa signatures: An mpc approach," *Cryptology ePrint Archive*, 2025.

[48] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-dilithium: A lattice-based digital signature scheme," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 238–268, 2018.

[49] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Annual cryptology conference*. Springer, 2012, pp. 643–662.

[50] X. Wang, S. Ranellucci, and J. Katz, "Global-scale secure multiparty computation," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 39–56.

[51] D. Rotaru and T. Wood, "Marbled circuits: Mixing arithmetic and boolean circuits with active security," in *International Conference on Cryptology in India*. Springer, 2019, pp. 227–249.

[52] D. Demmler, T. Schneider, and M. Zohner, "Aby-a framework for efficient mixed-protocol secure two-party computation." in *NDSS*, 2015.

[53] R. Garg, K. Yang, J. Katz, and X. Wang, "Scalable mixed-mode mpc," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 523–541.

[54] S. Agrawal, D. Stehlé, and A. Yadav, "Round-optimal lattice-based threshold signatures, revisited," *Cryptology ePrint Archive*, 2022.

[55] YosysHQ GmbH, "YosysHQ: Open Source EDA Tools and Services," https://www.yosyshq.com/, accessed: 2025-08-19.

[56] D. Unruh, "Universally composable quantum multi-party computation," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2010, pp. 486–505.

[57] R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias, "Semi-homomorphic encryption and multiparty computation," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2011, pp. 169–188.

[58] Y. Zhou, W. Wang, Y. Sun, and Y. Yu, "Rejected challenges pose new challenges: Key recovery of crystals-dilithium via side-channel attacks," *Cryptology ePrint Archive*, 2025.

[59] Z. Liu, A. Wang, C. Wei, Y. Ding, J. Zhang, A. Liu, and L. Zhu, "Release the power of rejected signatures: An efficient side-channel attack on dilithium," *Cryptology ePrint Archive*, 2025.

[60] P. Azevedo-Oliveira, A. Calle Viera, B. Cogliati, and L. Goubin, "Finding a polytope: A practical fault attack against dilithium," in *IACR International Conference on Public-Key Cryptography*. Springer, 2025, pp. 259–283.

[61] x509-parser Developers (Rusticata), "Tbscertificate struct," https://docs.rs/x509-parser/latest/x509_parser/certificate/struct.TbsCertificate.html, Rust crate x509-parser, 2025, accessed: 2025-08-26.

[62] V. 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, 2009, pp. 598–616.

[63] R. Cramer, I. Damgård, D. Escudero, P. Scholl, and C. Xing, "Spdz2k: Efficient mpc mod 2k for dishonest majority," in *Annual International Cryptology Conference. Springer*, 2018, pp. 769–798.

[64] A. Aly, K. Cong, D. Cozzo, M. Keller, E. Orsini, D. Rotaru, O. Scherer, P. Scholl, N. P. Smart, and T. Tanguy, "Scale-mamba v1. 14: Documentation," 2021.

[65] D. Cozzo and N. P. Smart, "Sharing the luov: threshold post-quantum signatures," in *IMA International Conference on Cryptography and Coding*. Springer, 2019, pp. 128–153.

[66] M. J. Dworkin *et al.*, "Sha-3 standard: Permutation-based hash and extendable-output functions," 2015.

[67] A. Kyster, F. H. Nielsen, S. Oechsner, and P. Scholl, "Rushing at spdz: On the practical security of malicious mpc implementations," in *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2025, pp. 2491–2508.

[68] R. Cramer, I. Damgård, D. Escudero, P. Scholl, and C. Xing, "Spd$\mathbb{Z}_{2^k}$: Efficient mpc mod 2^k for dishonest majority," in *Advances in Cryptology – CRYPTO 2018*, ser. Lecture Notes in Computer Science, vol. 10992. Springer, 2018, pp. 769–798.

[69] X. Wang, A. J. Malozemoff, and J. Katz, "EMP-toolkit: Efficient MultiParty Computation Toolkit," https://github.com/emp-toolkit, 2016.

[70] IDManagement, "Federal pki governance and compliance audit information," https://www.idmanagement.gov/fpki/notifications/#fpki-graph, 2025, [Accessed 2025-08-10].

[71] H. Qiu, M. Qiu, and R. Lu, "Secure v2x communication network based on intelligent pki and edge computing," *IEEE Network*, vol. 34, no. 2, pp. 172–178, 2019.

[72] Federal Public Key Infrastructure Management Authority (FPKIMA), "Federal public key infrastructure (fpki) trust infrastructure certification practice statement (cps)," U.S. General Services Administration, Tech. Rep. Version 6.0, Jun. 28 2021, includes practices for FBCA and FCPCA, aligned with RFC 3647. [Online]. Available: https://www.idmanagement.gov/docs/archived/fpki-fpkima-cps-v60.pdf

[73] U.S. Department of State PKI Policy Management Authority, "Public Key Infrastructure (PKI) X.509 Certificate Policy (CP), Version 2.1," U.S. Department of State, Tech. Rep., Sep. 30 2023, version 2.1; approved by the PKI Policy Management Authority; defines assurance levels, cross-certification with FBCA, and aligns with RFC 3647. [Online]. Available: https://pkaps.pki.state.gov/docs/DOSPKIX.509CPv2.1(20230830)Signed.pdf

[74] D. Boneh, R. Gennaro, S. Goldfeder, A. Jain, S. Kim, P. M. Rasmussen, and A. Sahai, "Threshold cryptosystems from threshold fully homomorphic encryption," in *Annual International Cryptology Conference*. Springer, 2018, pp. 565–596.

[75] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE, 2001, pp. 136–145.

[76] R. Bacho, S. Das, J. Loss, and L. Ren, "Glacius: threshold schnorr signatures from ddh with full adaptive security," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2025, pp. 304–334.

[77] C. Komlo and I. Goldberg, "Arctic: Lightweight and stateless threshold schnorr signatures," in *IACR International Conference on Public-Key Cryptography*. Springer, 2025, pp. 234–267.

[78] F. Benhamouda, S. Halevi, H. Krawczyk, Y. Ma, and T. Rabin, "Sprint: High-throughput robust distributed schnorr signatures," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2024, pp. 62–91.

[79] Y. Lyu, Z. Li, H.-S. Zhou, H. Xue, M. Wang, S. Wang, and M. Liu, "Bandwidth-efficient robust threshold ecdsa in three rounds," *Cryptology ePrint Archive*, 2025.

[80] G. Tang, S. Han, L. Lin, C. Wei, and Y. Yan, "Batch range proof: How to make threshold ecdsa more efficient," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 4256–4270.

[81] Y. Xie, Q. Fan, C. Zhang, T. Wu, Y. Zhou, D. He, and L. Zhu, "Accountable and secure threshold eddsa signature and its applications," *IEEE Transactions on Information Forensics and Security*, 2024.

[82] Q. Feng, K. Yang, K. Zhang, X. Wang, Y. Yu, and X. Xie, "Stateless deterministic multi-party eddsa signatures with low communication," in *IACR International Conference on Public-Key Cryptography*. Springer, 2025, pp. 268–297.

[83] L. De Feo and M. Meyer, "Threshold schemes from isogeny assumptions," in *IACR International Conference on Public-Key Cryptography*. Springer, 2020, pp. 187–212.

[84] J. Devevey, A. Passelègue, and D. Stehlé, "G+ g: a fiat-shamir lattice signature based on convolved gaussians," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2023, pp. 37–64.

[85] I. Damgård, C. Orlandi, A. Takahashi, and M. Tibouchi, "Two-round n-out-of-n and multi-signatures and trapdoor commitment from lattices," in *IACR International Conference on Public-Key Cryptography*. Springer, 2021, pp. 99–130.

[86] N. A. Alkadri, N. Döttling, and S. Pu, "Practical lattice-based distributed signatures for a small number of signers," in *International Conference on Applied Cryptography and Network Security*. Springer, 2024, pp. 376–402.

[87] R. Del Pino, S. Katsumata, M. Maller, F. Mouhartem, T. Prest, and M.-J. Saarinen, "Threshold raccoon: practical threshold signatures from standard lattice assumptions," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2024, pp. 219–248.

[88] S. Katsumata, M. Reichle, and K. Takemure, "Adaptively secure 5 round threshold signatures from mlwe/msis and dl with rewinding," in *Annual International Cryptology Conference*. Springer, 2024, pp. 459–491.

[89] C. Zhu and S. Tessaro, "The algebraic one-more misis problem and applications to threshold signatures," *Cryptology ePrint Archive*, 2025.

[90] J. Massimo, P. Kampanakis, S. Turner, and B. E. Westerbaan, "Internet X.509 Public Key Infrastructure – Algorithm Identifiers for the Module-Lattice-Based Digital Signature Algorithm (ML-DSA)," RFC 9881, Internet Engineering Task Force (IETF), Standards Track, October 2025, status: Standards Track. [Online]. Available: https://www.rfc-editor.org/info/rfc9881

Appendix A. Intractability Assumptions

Definition A.1 (Module LWE (MLWE))**.** Let $\mathsf{pp} = (N, k, \ell, q, \eta)$, where N, k, ℓ, q, η are positive integers. We say that MLWE holds w.r.t. pp if for every PPT algorithm $\mathcal{A}$ the advantage $\mathsf{Adv}^{\mathsf{MLWE}}_{\mathcal{A}}(\mathsf{pp})$ is negligible in λ, where

$$\mathsf{Adv}^{\mathsf{MLWE}}_{\mathcal{A}}(\mathsf{pp}) := \left| \begin{array}{l} \Pr\left[\begin{array}{l} \mathbf{A} \leftarrow_{\$} R_q^{k\times\ell},\ \mathbf{s} \leftarrow_{\$} S_\eta^{k+\ell}; \\ \mathbf{t} := [I_k \mid \mathbf{A}] \cdot \mathbf{s} \in R_q^k; \\ b \leftarrow \mathcal{A}(\mathsf{pp}, \mathbf{A}, \mathbf{t}) :\ b = 1 \end{array}\right] \\ -\Pr\left[\begin{array}{l} \mathbf{A} \leftarrow_{\$} R_q^{k\times\ell},\ \mathbf{t} \leftarrow_{\$} R_q^k; \\ b \leftarrow \mathcal{A}(\mathsf{pp}, \mathbf{A}, \mathbf{t}) :\ b = 1 \end{array}\right] \end{array} \right|$$

Definition A.2 (Module SIS (MSIS))**.** Let $\mathsf{pp} = (N, k, \ell, q, \beta)$, where N, k, ℓ, q are positive integers and β is a positive real. We say that MSIS holds w.r.t. pp if for every algorithm $\mathcal{A}$ the advantage $\mathsf{Adv}^{\mathsf{MSIS}}_{\mathcal{A}}(\mathsf{pp})$ is negligible in λ, where

$$\mathsf{Adv}^{\mathsf{MSIS}}_{\mathcal{A}}(\mathsf{pp}) := \Pr\left[\begin{array}{c} 0<\|\mathbf{x}\|\le\beta \ \wedge\ [I_k|\mathbf{A}]\cdot\mathbf{x}\equiv 0 \pmod q : \\ \mathbf{A}\leftarrow_{\$} R_q^{k\times\ell},\ \mathbf{x}\leftarrow\mathcal{A}(\mathsf{pp},\mathbf{A}) \end{array}\right]$$

Definition A.3 (SelfTargetMSIS)**.** Let k, ℓ, q be integers and $B_{\mathsf{stmsis}} > 0$ be a real number. Let $\mathcal{C}$ be a subset of R_q, and let $G : R_q^k \times \{0,1\}^{2\kappa} \to \mathcal{C}$ be a cryptographic hash function modeled as a random oracle. The advantage $\mathsf{Adv}^{\mathsf{SelfTargetMSIS}}_{\mathcal{A}}(\kappa)$ of an adversary $\mathcal{A}$ against $\mathsf{SelfTargetMSIS}_{q,k,\ell,B_{\mathsf{stmsis}}}$ is defined as:

$$\mathsf{Adv}^{\mathsf{SelfTargetMSIS}}_{\mathcal{A}}(\kappa) = \Pr\left[\begin{array}{l} \mathbf{A} \leftarrow_{\$} R_q^{k\times\ell},\ (\mathsf{msg}, \mathbf{z}) \leftarrow \mathcal{A}^G(\mathbf{A}), \\ (\mathsf{msg}, \mathbf{z}) \in \{0,1\}^{2\kappa} \times R_q^{k+\ell}, \\ \text{where } \mathbf{z} = \begin{bmatrix} \mathbf{z}' \\ \mathbf{c} \end{bmatrix},\ \|\mathbf{z}\|_2 \le B_{\mathsf{stmsis}}, \\ G([\mathbf{I} \mid \mathbf{A}] \cdot \mathbf{z}, \mathsf{msg}) = c \end{array}\right]$$

The $\mathsf{SelfTargetMSIS}_{q,k,\ell,\beta}$ assumption states that any efficient adversary $\mathcal{A}$ has a negligible advantage $\mathsf{Adv}^{\mathsf{SelfTargetMSIS}}_{\mathcal{A}}(\kappa)$.

Appendix B. Core Components of ML-DSA

In section 3, we outlined the end-to-end ML-DSA workflow, from key generation to verification, and highlighted its core subroutines, including HighBits and SampleInBall. We present these components in Figure 4.

Appendix C. Garbled Circuit Gadgets

Table 9 summarizes all functionalities and highlights their complexity when thresholded using the WRK17 protocol, consistent with the analysis in section 4. The table is organized into two layers. The first layer lists generic primitives, such as the Keccak-f permutation used in SHAKE operations (*e.g.,* absorb, squeeze, squeezeblocks), and the 23 bit unsigned adder used in A2B share conversion. The second layer presents ML-DSA-specific gadgets. For each function, we provide a gate-level breakdown along with the number and bit length of inputs and outputs. Some gadgets, including MakeHint and Decompose, use the same circuit for NIST security levels 3 and 5, while others, such as FisherYates and CheckNorm, maintain consistent input and output formats across levels. As discussed in section 4, these functionalities are combined in the implementation to reduce round complexity, which is crucial in high-latency networks.

Appendix D. Trusted Key Generation and MPC Setup

This section summarizes the setup-phase runtime overhead, encompassing ML-DSA key generation, private key sharing and distribution, correlated randomness provisioning, and WRK17 circuit preprocessing. We report the amortized cost, including both online time and preprocessing time for SPDZ and WRK17 in Table 10.

ML-DSA Key Generation. A trusted dealer generates ML-DSA key pairs $(\mathsf{sk}, \mathsf{pk})$ independently of the number of

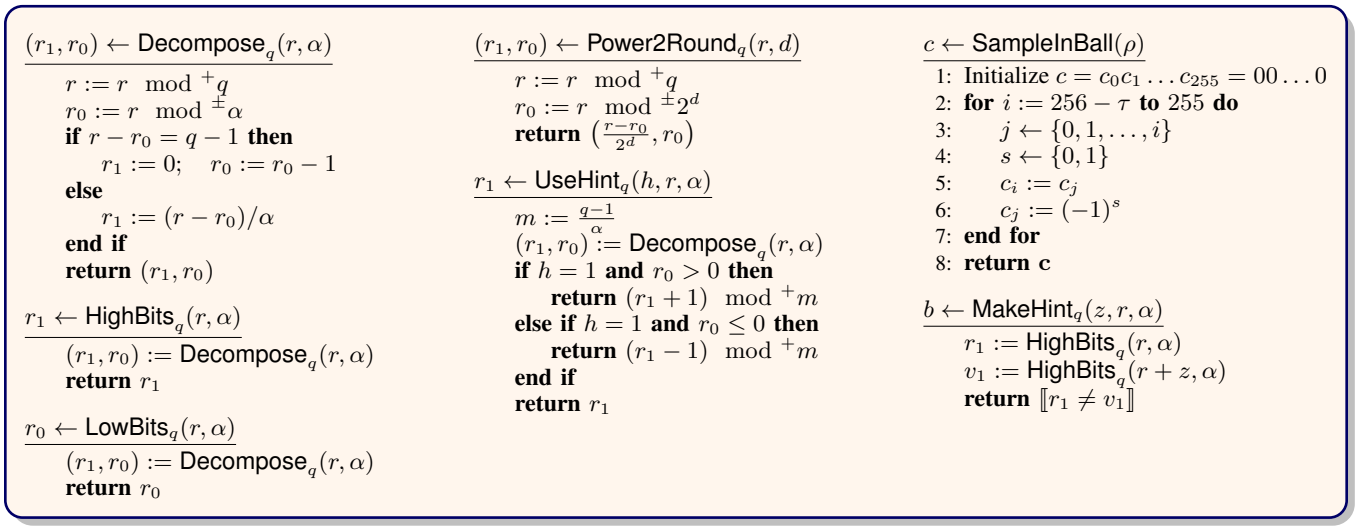

Figure 4: FIPS-204 core supporting components

servers. The measured generation times are 70 μs, 117 μs, and 180 μs for ML-DSA-44, ML-DSA-65, and ML-DSA-87, respectively.

Private Key Sharing and Distribution. After generation, the dealer secret shares the private key components $\mathbf{s}_1$, $\mathbf{s}_2$, and $\mathbf{t}_0$ at the coefficient level following the NTT transform, as discussed in section 4, while the secret key K is shared byte-wise. The distribution time for these shares ranges from 5.9-19.3 ms for ML-DSA-44, 8.3-25.1 ms for ML-DSA-65, and 11.2-32.2 ms for ML-DSA-87 when scaling from 2 to 5 parties.

Correlated Randomness Provisioning. Finally, the trusted dealer generates CRVs, including daBits, Beaver triples, and AGVs, and distributes them securely to the servers. The number of CRVs required depends on the chosen ML-DSA parameter set, as detailed in Table 7. This step requires between 1-14 s, during which approximately 13K-50K CRVs are generated for ML-DSA security levels 2 through 5.

WRK17 Thresholded Functionalities. In the WRK17 preprocessing model, the setup proceeds in two stages. The function-independent phase generates random AND triples authenticated with IT-MACs, independent of the target computation. In the subsequent function-dependent phase, these triples are consumed to construct a single authenticated garbled table specific to the target circuit. This separation shifts the cryptographic workload to the offline stage, leaving only lightweight evaluation during the online phase. The measured preprocessing costs (for low latency network) for functionalities mapped to WRK17 are reported in Table 11 per our analysis in section 4. In this table, Keccak-f and Mod23Add are independent of ML-DSA parameters, whereas other operations are either parameter-specific (*e.g.,* FisherYates) or shared across security levels (*e.g.,* Decompose for ML-DSA-65/87). We reiterate, as discussed in section 4, that these functionalities are combined in the implementation to reduce round complexity. This combination reduces input wires and eliminates output wires for intermediate values. Moreover, the number of nonlinear gates is roughly the sum of those from each component. Thus, the resulting reduction in wires yields improved overall preprocessing time for SHIELD in the implementation.

TABLE 10: Amortized cost (min) for one successful ML-DSA signing iteration.

Algorithm	Latency Setting	# Servers			
		2	3	4	5
SHIELD *(NIST-2)*	Low	2.806	4.482	6.690	8.607
	Med	7.982	12.751	19.037	24.491
	High	23.726	37.904	56.590	72.806
SHIELD *(NIST-3)*	Low	3.727	5.950	8.891	11.365
	Med	10.602	16.928	25.299	32.339
	High	31.513	50.319	75.205	96.135
SHIELD *(NIST-5)*	Low	4.969	7.914	11.827	15.107
	Med	14.134	22.516	33.653	42.988
	High	42.011	66.929	100.039	127.793

TABLE 9: Detailed Gate-Level Overview of Gadget Functionalities in Garbled Circuits

Function	Used in Algorithm	Gate Summary			Input (bits) (First / Second)	Output (bits) (First / Second)
		XOR ($\oplus$)	AND ($\wedge$)	INV ($\neg$)		
Keccak-f	SHAKE Family	115200	38400	38486	1600	1600
Mod23Add	A2Y	68	122	96	23 / 23	23
Decompose	ML-DSA-44	207	424	357	23	6 / 23
	ML-DSA-65 / ML-DSA-87	75	127	105	23	4 / 23
FisherYates	ML-DSA-44	0	223337	131774	1088	512
	ML-DSA-65	0	279127	163939		
	ML-DSA-87	0	337571	198426		
MakeHint	ML-DSA-44	4253	24197	17616	5888 / 1536	264
	ML-DSA-65 / ML-DSA-87	4753	25160	16266	5888 / 1024	264
HintOneCmp	ML-DSA-44	0	10	14	11	1
	ML-DSA-65	0	7	7		
	ML-DSA-87	0	8	10		
CheckNorm	ML-DSA-44 (γ_2)	5738	35079	22890	5888	1
	ML-DSA-44 ($\gamma_2 - \beta$)	5596	34845	24115		
	ML-DSA-44 ($\gamma_1 - \beta$)	5738	35079	22890		
	ML-DSA-65 (γ_2)	1240	25772	16647		
	ML-DSA-65 ($\gamma_2 - \beta$)	5895	35561	23364		
	ML-DSA-65 ($\gamma_1 - \beta$)	7105	36771	24024		
	ML-DSA-87 (γ_2)	1240	25772	16647		
	ML-DSA-87 ($\gamma_2 - \beta$)	5950	34269	23383		
	ML-DSA-87 ($\gamma_1 - \beta$)	6382	34492	22418		

Circuit Input and Output: Some circuits may have two inputs/outputs. If so, they are given as First/Second. If only one, it appears as a single value.
CheckNorm: This gadget evaluates whether an input (polynomial coefficient) satisfies one of three predefined norm bounds: γ_2, $\gamma_2 - \beta$, and $\gamma_1 - \beta$. These bounds are hard-coded into the circuit logic rather than provided dynamically as inputs, resulting in nine distinct circuit instances across the supported security levels.

TABLE 11: Per signer preprocessing runtime (ms) for WRK17-thresholded functionalities.

Mode	Operation	# servers			
		2	3	4	5
–	Keccak-f	126.0	203.0	289.0	369.0
	Mod23Add	1.51	2.39	3.67	4.69
2	Decompose	1.85	3.12	4.62	6.21
	CheckNorm($\gamma_1 - \beta$)	90.0	145.0	210.0	285.0
	CheckNorm($\gamma_2 - \beta$)	85.0	155.0	205.0	288.0
	CheckNorm(γ_2)	60.0	100.0	153.0	215.0
	FisherYates	580.0	930.0	1340.0	1870.0
	MakeHint	230.0	400.0	620.0	850.0
3	CheckNorm($\gamma_1 - \beta$)	128.0	195.0	273.0	343.0
	CheckNorm($\gamma_2 - \beta$)	133.0	240.0	315.0	413.0
	CheckNorm(γ_2)	98.0	165.0	228.0	293.0
	FisherYates	700.0	1180.0	1680.0	2170.0
5	CheckNorm($\gamma_1 - \beta$)	150.0	258.0	365.0	473.0
	CheckNorm($\gamma_2 - \beta$)	180.0	288.0	395.0	503.0
	CheckNorm(γ_2)	135.0	233.0	330.0	428.0
	FisherYates	690.0	1100.0	1510.0	1910.0
35	Decompose	1.54	2.48	4.01	5.16
	MakeHint	380.0	660.0	960.0	1250.0

HintOneCmp: The preprocessing time is same across security levels (total: 1.41 ms).

Appendix E. Meta-Review

The following meta-review was prepared by the program committee for the 2026 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

E.1. Summary

This paper presents SHIELD, a threshold signing framework for ML-DSA that follows the FIPS-204 signing algorithm without modification while producing standard ML-DSA signatures. The system targets distributed CA signing in the U.S. Federal PKI and includes an open-source implementation with performance measurements under varied network latency.

E.2. Scientific Contributions

- Creates a New Tool to Enable Future Science
- Addresses a Long-Known Issue
- Provides a Valuable Step Forward in an Established Field

E.3. Reasons for Acceptance

1) The paper provides a valuable step forward in an established field by demonstrating that a practically usable threshold ML-DSA can be engineered while preserving standard verification and strict algorithmic adherence to FIPS-204 signing steps.
2) The paper creates a new tool to enable future science by releasing an open-source implementation that others can audit, reproduce, and extend for regulated PKI deployments.
3) The paper addresses a long-known issue by reducing single-key compromise risk for CA signing operations without requiring changes to the verification ecosystem, and it provides empirical evidence that the approach is feasible for PKI-style workloads.