

# High-Throughput Post-Quantum Signature Framework for Time-Critical Communications

Lazar Lazarevic and Attila A. Yavuz  
University of South Florida, Tampa, FL 33620, USA  
Email: {lazarlazarevic, attilaayavuz}@usf.edu

**Abstract**—High-throughput, time-critical communication systems such as vehicular networks authenticating hundreds of Basic Safety Messages per second, continuous satellite telemetry streams, and 5G broadcast services demand digital authentication at rates that challenge even conventional cryptography. Across these settings, a viable signature scheme must offer massive signing throughput, exceptionally low end-to-end delay, and a compact payload to prevent network congestion on constrained links. While lattice-based algorithms, such as the recently standardized NIST-PQC candidates, offer the most balanced trade-offs among all post-quantum alternatives, they are hindered by computationally expensive signing operations and relatively large payloads. More critically, existing mechanisms fail to simultaneously exploit their inherent offline-capable structures and highly parallelizable properties.

To overcome this fundamental bottleneck, we propose **SOKO**, which is, to the best of our knowledge, the first full-pipeline GPU-accelerated online/offline post-quantum signature framework tailored for lattice-based hash-and-sign algorithms. **SOKO** harnesses the message-independent phase of **FN-DSA** to mass-generate cryptographic pre-tokens via batched, NTT-based pre-processing on the GPU. Experimental results demonstrate that **SOKO** achieves a cryptographic end-to-end delay of 2.3  $\mu$ s and a throughput of 433,537 signatures per second. This remarkable performance corresponds to a  $115\times$  improvement over **ML-DSA** and a  $69\times$  improvement over standard **FN-DSA** in both effective delay and signing throughput. Ultimately, **SOKO** demonstrates substantial cryptographic headroom for aggregated, bursty, and high-throughput workloads, and we open-source our complete framework for public testing and adaptation.

**Index Terms**—Post-quantum security; Time-Critical Communications; Real-Time Authentication; GPU Acceleration.

## I. INTRODUCTION

High-throughput, time-critical communication systems demand digital authentication at rates that challenge even conventional cryptographic schemes. In vehicular networks, roadside units and fleet coordinators must authenticate hundreds of Basic Safety Messages per second from surrounding vehicles to enable safe platooning and collision avoidance [1]. Satellite telemetry and imaging platforms must sign continuous high-rate data streams under strict latency budgets, with authentication requirements reaching thousands of frames per second. 5G Multicast and Broadcast Services (MBS) support point-to-multipoint delivery for group communication, public safety, IoT, and V2X applications, where a single signer authenticates a broadcast stream consumed by thousands of receivers simultaneously [2]. Across all these settings, three requirements dominate: (i) high signing throughput at the

sender, (ii) minimum cryptographic payload to limit fragmentation and retransmissions on constrained or lossy links, and (iii) fast per-message verification at every receiver.

Even classical signatures may be insufficient in certain high-rate authentication use cases. *ECDSA* and *Ed25519*, despite their compactness, achieve at most 14,749 signatures/s and incur end-to-end delays exceeding 65  $\mu$ s in our evaluation, which can be limiting for high-rate broadcast authentication scenarios. The transition to post-quantum cryptography (PQC) significantly worsens this cost model, but it is unavoidable for long-term authentication security. Shor’s algorithm showed that a sufficiently powerful quantum computer can solve integer factorization and discrete logarithms in polynomial time, directly threatening classical public-key cryptosystems such as RSA, DSA, ECDSA, and related elliptic-curve constructions [3]. This threat has motivated major standardization and policy efforts: NIST has standardized *ML-DSA* and *SLH-DSA* and is finalizing *FN-DSA*, while U.S. government directives and legislation call for migration toward quantum-resistant cryptography [4], [5], [6], [7]. Below, we summarize the limits of the state of the art most relevant to our work and show why PQ authentication must be evaluated as a system-level component that jointly affects signing throughput, payload size, and end-to-end delay.

### A. Related Work and Limitations

**The High Cost of PQ Signatures.** NIST-standardized PQ signature schemes introduce signing latencies and authentication payload sizes that create deployment challenges in latency-sensitive systems. *ML-DSA* incurs 262  $\mu$ s signing latency with 2,420-byte signatures, while *FN-DSA* reduces the signature size to 845 bytes but still requires 157  $\mu$ s per signature without preprocessing [4], [5], [6]. As our experiments show, standard *FN-DSA* achieves 6,317 signatures/s and *ML-DSA* reaches only 3,762 signatures/s, which is below the throughput target of high-rate broadcast authentication. Lightweight signature systems have also been explored for constrained and cloud-assisted embedded IoT environments, including *SCRA*-style approaches that reduce local signing and key-management costs for resource-limited devices [10]. In contrast, this work focuses on NIST-PQC signatures and their system-level enhancement because standards-compliant migration is critical for practical deployment and is em-

phasized by current PQC standardization and government transition efforts [4], [5], [6], [7].

PQ authentication has also been studied in TLS and X.509 certificate-chain settings. Prior PQ-TLS studies show that larger PQ certificates and slower signature operations can increase handshake latency and reduce server throughput [9]. These studies motivate the need to reduce PQ authentication overhead, but they primarily focus on interactive TLS-style handshakes rather than high-throughput broadcast authentication where a signer must continuously generate a large number of signatures.

#### Online/Offline and GPU Approaches in Isolation.

Online/offline (OO) signatures reduce latency by shifting message-independent work to a precomputation phase. This concept has been studied extensively for conventional signatures, starting from early online/offline signature constructions and later improved schemes that reduce the cost of the online signing phase [11], [12]. In the PQ setting, however, online/offline signing remains comparatively less explored. Only a small number of PQ-specific efforts consider OO transformations, including lattice-based OO constructions analyzed by Albrecht et al. and application-specific use in full-stack private federated learning [13], [14]. These works show that precomputation can reduce the online latency of PQ signatures, but they do not provide a GPU-accelerated token-buffered architecture for sustaining high-rate signing.

Beyond online/offline design, another path to improving PQ signature performance is hardware acceleration. GPU acceleration has been explored separately for Falcon [15], [16] and *FN-DSA*/Mitaka [17], with *GOLF* demonstrating throughput gains for the signing primitive itself. However, these efforts primarily accelerate the underlying primitive and do not integrate online/offline token generation, compact *FN-DSA*-based authentication, and CPU/GPU producer-consumer scheduling into one delay-aware authentication pipeline.

*No existing solution provides a delay-aware PQ signature framework that jointly combines online/offline signing, fast verification, compact authentication material, and hardware acceleration for high-throughput broadcast authentication. SOKO addresses this gap.*

#### B. Our Contributions

We present **SOKO**, the first comprehensive GPU-accelerated online/offline PQ authentication framework that integrates *FN-DSA*-based online/offline signing, compact authentication material, GPU token generation, and CPU-side low-latency signing into a unified high-throughput pipeline. **SOKO** assigns message-independent token generation to the GPU and keeps latency-sensitive online signing on the CPU. The GPU continuously produces one-time tokens via batched NTT-based preprocessing, while CPU signing threads consume tokens from a host-side ring buffer as messages arrive. This producer-consumer design eliminates GPU kernel launch and PCIe transfer overhead from the real-time signing path, enabling high-throughput PQ broadcast authentication with minimal online latency.

The contributions of this work are as follows:

- *GPU-Accelerated Offline Pipeline.* We design a CUDA-based offline token-generation pipeline using NTT-based convolution, per-block parallel computation, asynchronous D2H transfers, and pinned host-memory banks. This reduces amortized offline cost to  $0.5 \mu\text{s}$  per token, an  $85\times$  reduction over single-core CPU and  $4.6\times$  over multi-threaded CPU.
- *Token-Buffered Signing Architecture.* We develop a fixed-size ring-buffer queue with a threshold-based refill policy that decouples GPU offline preprocessing from CPU online signing, preventing token starvation while sustaining concurrent production and consumption. This achieves a  $142.7\times$  presign throughput improvement over single-core *OO-FN-DSA*.
- *System-Level Performance.* **SOKO** achieves 433,537 signatures/s and  $2.3 \mu\text{s}$  cryptographic end-to-end delay, a  $115\times$  improvement over *ML-DSA* and a  $69\times$  improvement over standard *FN-DSA*, the highest throughput and lowest delay among all evaluated PQ constructions.
- *Implementation.* A complete implementation is available at: <https://github.com/LazarevicLazar/SOKO>.

## II. PRELIMINARIES

**PQ Signatures:** A digital signature scheme consists of *KeyGen*, *Sign*, and *Verify*, which generate a key pair  $(pk, sk)$ , sign a message  $M$ , and verify a signature  $\sigma$ , respectively. In this work, we focus on PQ signatures for high-throughput authentication, with emphasis on lattice-based schemes such as *ML-DSA* and *FN-DSA*.

**Notation and Metrics:** For a signature scheme  $A$ , we use  $|pk_A|$  and  $|\sigma_A|$  to denote public-key and signature sizes when discussing authentication payload. We use  $T_{\text{off}}$ ,  $T_{\text{on}}$ , and  $T_{\text{ver}}$  for offline preprocessing, online signing, and verification latency. Cryptographic end-to-end delay is defined as  $T_{\text{on}} + T_{\text{ver}}$ , excluding network propagation and application-layer processing. Signing throughput is reported in signatures per second, and  $N$  denotes the number of offline tokens in one token bank.

***FN-DSA* and Online/Offline Transformation:** *FN-DSA* is based on Falcon, a compact lattice-based PQ digital signature scheme selected by NIST for standardization [8], [6]. Its signing samples short lattice vectors relative to a message-derived target. In standard *FN-DSA*, this target is computed from  $\text{HashToPoint}(r \parallel \mu, q, n)$  [8], where  $r$  is a random nonce,  $\mu$  is the message digest,  $q$  is the modulus, and  $n$  is the polynomial dimension. Because the target is message-dependent, sampling remains on the online signing path.

Online/offline *FN-DSA* introduces a message-independent preprocessing phase [13]. During the offline phase, two short lattice vectors  $(u_0, u_1)$  are sampled and combined with the public key  $h$  to construct  $c_{\text{pre}} = u_0 - hu_1 \pmod{q}$ . The tuple  $(u_0, u_1, c_{\text{pre}})$  forms a one-time offline token. Once the message arrives, the online phase computes the message-dependent hash target, combines it with  $c_{\text{pre}}$ , removes the offline mask, and outputs a valid *FN-DSA* signature. This

shifts expensive computation outside the latency-critical authentication path.

**Polynomial Transforms:** Lattice-based signatures rely on polynomial arithmetic. FFT operations are used by the *FN-DSA* sampler, while NTT operations provide efficient modular polynomial convolution. Since NTTs consist of regular butterfly stages and pointwise modular arithmetic, they map efficiently onto GPU execution [15], [16], [17]. **SOKO** uses this property to accelerate offline token generation.

### III. SYSTEM ARCHITECTURE

#### A. System Model

The proposed system consists of two main entities: (i) a resourceful broadcast signer with CPU and GPU resources, and (ii) receiver entities that obtain signed messages and verify the client signature before accepting the payload. The signer is the performance-critical entity because it may generate a large number of signatures per second under time-critical communication constraints. Receivers may be resourceful gateways or constrained field devices depending on the deployment setting. Messages are delivered in a broadcast setting, where one signed message may be verified by many receivers [2].

#### B. Design Requirements and Justifications

The design of **SOKO** is driven by three system-level requirements: high signing throughput, compact authentication material, and low end-to-end delay. In the target broadcast setting, a single signer may generate many signatures per second while each signed message may be verified by many receivers. Therefore, the signing path must be minimized, the cryptographic payload should remain compact, and verification must remain fast enough for receiver-side scalability.

**Enabling High-Throughput Signer Performance:** Lattice-based signature schemes provide the most balanced trade-off between signing performance and cryptographic overhead among practical PQ candidates. In particular, *ML-DSA* and *FN-DSA* rely primarily on polynomial multiplication, modular reduction, and vector transforms, operations which map efficiently onto massively parallel hardware architectures and enable GPU acceleration [4], [6], [8], [15], [16], [17].

To further reduce online signing latency, online/offline transformations shift expensive cryptographic operations into an offline preprocessing stage while minimizing computation during the online signing phase [11], [12], [13]. While these techniques are well-established for conventional signatures, there are currently few online/offline PQ alternatives available.

Among lattice-based constructions, *OO-ML-DSA* and *OO-FN-DSA* provide the most favorable trade-offs. *OO-ML-DSA* achieves low online signing latency ( $\sim 65 \mu\text{s}$ ), but incurs large offline preprocessing cost and 2420-byte signatures [14]. *OO-FN-DSA* improves both computational and communication performance with  $\sim 22 \mu\text{s}$  online signing latency, 845-byte signatures, 897-byte public keys [13], and  $\sim 19 \mu\text{s}$  verification latency. As a result, *OO-FN-DSA* provides the most

balanced trade-off between signing throughput, verification performance, and transmission efficiency for integration into **SOKO**.

**Payload Size and End-to-End Delay:** PQ signatures increase the authentication material transmitted with each signed message. In high-throughput broadcast settings, larger signatures increase bandwidth use, packetization pressure, and retransmission costs on constrained or lossy links. *FN-DSA* is attractive in this setting because it provides substantially smaller authentication material than *ML-DSA*, with 845-byte signatures and 897-byte public keys, while maintaining fast verification [4], [6], [8]. End-to-end delay is therefore determined by the combined cost of online signing, message transmission, and receiver-side verification. Since one broadcast message may be verified by many receivers, **SOKO** prioritizes low online signing latency, compact signatures, and fast verification as its main system-level objectives.

Therefore, *OO-FN-DSA* is selected as the core signing primitive because it combines low online signing latency, fast verification, compact signatures, and compatibility with parallel offline preprocessing.

#### C. Proposed Framework

Figure 1(b) details the computation performed inside one block. Thread  $j$  samples the values  $u_0(j)$  and  $u_1(j)$  using the PRNG parameterized by  $(seed, token, j)$  (1, L15 – L17). This produces deterministic but distinct per-thread samples for the current presign. These values pass through the butterfly stages of the NTT computation (2, L19 – L20). The transform is executed level by level, the butterfly stride doubles at each stage, and thread synchronization ensures that one level is completed before the next begins. This stage forms the fine-grained parallel computation inside each presign block.

After the butterfly network completes, the transformed intermediates are combined with the preloaded transformed public polynomial  $h$ , reduced modulo  $q$ , and then passed through the inverse transform to recover the coefficient-domain representation needed for the presign (3, L20 – L21). The algorithm computes the precomputed target  $c_{\text{pre}}$ , which is the token-dependent helper value later used during online signing (4, L21). Finally, it emits the completed offline token as  $\langle c_{\text{pre}}, u_0, u_1 \rangle$  (5, L22). A full kernel launch therefore produces a bank of such tokens.

Figure 1 shows the architecture of the proposed **SOKO** framework. The design separates expensive offline token generation from latency-sensitive online signing. The GPU is used to generate banks of presigns in parallel, while the CPU stores, schedules, and consumes these presigns to produce final signatures on demand.

The offline phase starts at the one-time CPU precomputation stage (a, L1 – L5). The host prepares the static transform-domain data required throughout token generation, including the transformed basis values  $B$  and the transformed public polynomial  $h$ . These values are reused across all subsequent token-generation batches, they are computed once and retained in host memory. The transformed value of  $h$  is

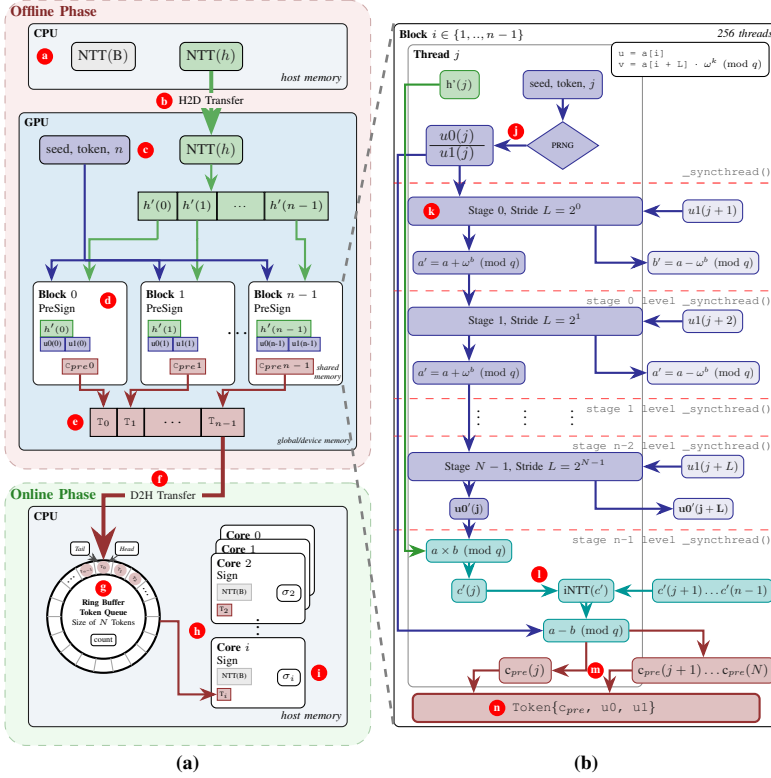


Fig. 1: System architecture of SOKO.

copied to the GPU through an H2D transfer so that it can be reused by all blocks during presign generation (b, L1, L13, L20).

The host launches the CUDA kernel with the tuple  $(seed, token, n)$ , where  $seed$  initializes the pseudorandom sampling procedure,  $token$  identifies the token stream, and  $n$  specifies the number of presigns to generate in the current bank (c, L8, L13). The kernel assigns one presign-generation task to each block (L14). Each block therefore receives the seed, token identifier, and batch parameters needed to generate one presign independently of the others (d, L14 – L22). The outputs are written to the device-side token bank (e, L22), which temporarily stores the generated presigns in GPU memory before they are transferred back to the host.

Once a token bank is ready, D2H transfers it from GPU memory back to host memory (f, L9). The CPU then inserts the received bank into the ring buffer token queue (g, L10). The **SOKO.RefillIfNeeded** routine checks the queue occupancy and launches a new GPU token-generation batch whenever the buffered token count falls below the refill threshold (L6 – L12). This queue bridges offline production and online consumption. It has a fixed size and maintains a head pointer, a tail pointer, and a counter. The head indicates the next token to be consumed, the tail indicates the next position into which a newly received token will be inserted, and the counter tracks the number of valid buffered tokens currently present in the queue. This organization allows GPU token generation and CPU token consumption to proceed asynchronously, while the

**Require:**  $sk = (f, g, F, G)$ ,  $pk = h$ ,  $n = 2^{\log n}$ ,  $q$ ,  $N$  tokens

**Precompute (CPU):**

- 1:  $\hat{B} \leftarrow \text{FFT}([g, -f; G, -F])$ ;  $\hat{H} \leftarrow \text{NTT}(h)$
- 2:  $t_{\text{fill}} \leftarrow \text{GPU bank generation time}$
- 3:  $t_{\text{sign}} \leftarrow \text{average signing time}$
- 4:  $\text{Threshold} \leftarrow \lceil 1.2 \cdot (t_{\text{fill}}/t_{\text{sign}}) \rceil$
- 5: Init buffer  $Q$  (size  $N$ ), Host banks  $U_1, U_2, C_{\text{pre}}$  size  $N \times n$
- 6: **procedure** **SOKO.REFILLIFNEEDED**
- 7:   **if**  $Q.\text{count} \leq \text{Threshold}$  **then**
- 8:      $seed \leftarrow \text{CSPRNG}()$ ; **Launch** **PRESIGN**
- 9:     **D2H copy**  $U_1, U_2, C_{\text{pre}}$ ;
- 10:    **Enqueue**  $[0 \dots N-1]$  into  $Q$
- 11:   **end if**
- 12: **end procedure**
- 13: **procedure** **SOKO.PRESIGN**( $\hat{H}, seed, n, U_1, U_2, C_{\text{pre}}$ )
- 14:   **for each block**  $i$  **in parallel do**
- 15:     **for each thread**  $j$  **in parallel do**
- 16:        $u_0[j] \leftarrow \text{GaussianSample}(\text{PRNG}(seed_1, i, j))$
- 17:        $u_1[j] \leftarrow \text{GaussianSample}(\text{PRNG}(seed_2, i, j))$
- 18:     **end for**
- 19:      $\bar{u}_{\text{synth}} \leftarrow \text{NTT}(u_1)$ ;  $\hat{C} \leftarrow \hat{H} \odot \hat{U}_1$
- 20:      $conv \leftarrow \text{iNTT}(\hat{C})$ ;  $c_{\text{pre}} \leftarrow (u_0 - conv) \bmod q$
- 21:     **write**  $U_1[i] = u_0, U_2[i] = u_1, C_{\text{pre}}[i] = c_{\text{pre}}$
- 22:   **end for**
- 23: **end procedure**
- 24: **function** **SOKO.SIGN**( $\mu$ )
- 25:   **SOKO.REFILLIFNEEDED**;  $i \leftarrow Q.\text{dequeue}()$
- 26:    $u_0 \leftarrow U_1[i]$ ;  $u_1 \leftarrow U_2[i]$ ;  $c_{\text{pre}} \leftarrow C_{\text{pre}}[i]$
- 27:    $r \leftarrow \{0, 1\}^{320}$ ;  $c \leftarrow \text{HashToPoint}(r \parallel \mu, q, n) + c_{\text{pre}}$
- 28:    $\hat{c} \leftarrow \text{FFT}(c)$
- 29:    $\hat{t} \leftarrow (-\frac{1}{q} \cdot \hat{c} \odot \text{FFT}(F), -\frac{1}{q} \cdot \hat{c} \odot \text{FFT}(f))$
- 30:    $t \leftarrow \text{round}(\text{invFFT}(\hat{t}))$ ;  $\hat{t} \leftarrow \text{FFT}(t)$
- 31:    $\hat{s} \leftarrow (\text{FFT}(c), \text{FFT}(0)) - \hat{t} \odot \hat{B}$
- 32:    $(s'_0, s'_1) \leftarrow \text{invFFT}(\hat{s})$
- 33:    $s_0 \leftarrow s'_0 - u_0$ ;  $s_1 \leftarrow s'_1 - u_1$
- 34:   **return**  $(r, \text{Compress}(s_1))$
- 35: **end function**
- 36: **end function**

Fig. 2: Algorithmic workflow of SOKO.

queue absorbs short-term rate mismatch between the producer and the consumers.

Tokens are dispatched from the ring buffer to the available CPU cores. Each core retrieves the next available token and applies it to a pending signing request (h, L26 – L27). The selected core performs the online completion of Falcon signing. Given a message and one token  $\langle c_{\text{pre}}, u_0, u_1 \rangle$ , the core executes the lightweight message-dependent computation and outputs the final signature (i, L28 – L35). Since the expensive sampling and transform operations have already been completed during the offline phase, the online computation is reduced, enabling low-latency signature generation.

#### D. Threat Model

We adopt the standard existential unforgeability under chosen-message attack (EUF-CMA) model for digital signatures [6], [13]. The adversary  $\mathcal{A}$  is a probabilistic polynomial-time (PPT) entity that receives the signer's public key  $pk$  and has adaptive access to a signing oracle  $\mathcal{O}_{\text{Sign}}$ .  $\mathcal{A}$  may observe all transmitted messages, signatures, and authentication material, and may mount active network attacks including replay, packet injection, substitution, and man-in-the-middle.  $\mathcal{A}$ 's goal is to produce an *existential forgery*: a pair  $(M^*, \sigma^*)$  such that  $\text{Verify}(pk, M^*, \sigma^*) = 1$  and  $M^*$  was never submitted to  $\mathcal{O}_{\text{Sign}}$ .

We assume the signing host provides trusted protection for the secret key, offline tokens, and CSPRNG state through OS isolation or hardware-backed protection. Host memory compromise and side-channel attacks are outside our threat model, consistent with the standard single-signer broadcast authentication setting [2]. Sustained demand above the refill rate may deplete the token buffer, causing signing to block and increase application latency rather than weaken security.

#### E. Security Analysis

We show that **SOKO**'s GPU-accelerated pipeline does not weaken the unforgeability guarantees of the underlying OO-FN-DSA construction [13]. The argument has two parts: (i) GPU-generated tokens are identically distributed to CPU-generated tokens, so the signing simulation is perfect, and (ii) the ring-buffer architecture enforces strict one-time token consumption.

**Theorem 1 (EUF-CMA Security of **SOKO**):** If *OO-FN-DSA* is *EUF-CMA* secure [13], then **SOKO** is *EUF-CMA* secure, provided that (i) the CSPRNG satisfies the pseudorandomness definition of [18], (ii) the ring buffer enforces one-time token consumption, and (iii) offline tokens are not disclosed from host memory.

**Proof Sketch:** Let  $\mathcal{A}$  be a PPT adversary against **SOKO** with non-negligible advantage  $\varepsilon$ . We construct a PPT reduction  $\mathcal{B}$  against *OO-FN-DSA*.  $\mathcal{B}$  receives  $pk$  and simulates **SOKO**'s signing oracle for  $\mathcal{A}$ . For each query  $M_i$ ,  $\mathcal{B}$  invokes **SOKO.PreSign** to obtain a fresh token  $T_i = (u_0^{(i)}, u_1^{(i)}, c_{pre}^{(i)})$  and returns  $\sigma_i = \text{SOKO.Sign}(M_i, T_i)$ . Because **SOKO.PreSign** on the GPU draws each coefficient of  $u_0, u_1$  from  $\mathcal{D}_{\mathbb{Z}^n, s}$  using the same sampler as *OO-FN-DSA.PreSign* on the CPU—differing only in that samples are derived from a fresh seed per launch and distinct (seed, token\_id,  $j$ ) tuples, preserving the intended token distribution. One-time use is enforced by the ring buffer's FIFO dequeue policy: once the head pointer advances past slot  $i$ , that slot is never re-inserted (Algorithm 1, lines 9–10), so no token is consumed twice. If  $\mathcal{A}$  outputs a forgery  $(M^*, \sigma^*)$ ,  $\mathcal{B}$  forwards it as a forgery against *OO-FN-DSA*, achieving advantage  $\varepsilon$ —a contradiction. ■

### IV. PERFORMANCE ANALYSIS

#### A. Experimental Setup

All experiments were conducted on Ubuntu 22.04 using an Intel Core i9-13900K CPU and an NVIDIA GeForce RTX 4070 GPU. CPU implementations were compiled with `gcc -O3 -Wall -Wextra -fomit-frame-pointer`, with OpenMP enabled using `-fopenmp` when applicable. GPU kernels were compiled with `nvcc 13.2.78` using `-O3` and Ada-generation CUDA targets (`compute_89/sm_89`). **SOKO** uses persistent CUDA contexts, asynchronous streams, pinned host memory, and asynchronous D2H transfers to reduce preprocessing and PCIe transfer overhead. Reported values are averaged execution times. In the plots, CPU and CPUMT denote single-threaded and multi-threaded CPU baselines.

TABLE I: Comparison of **SOKO** and Its Counterparts

| Algorithm         | PQ? | Offline ( $\mu$ s)     | Online ( $\mu$ s) | Verify ( $\mu$ s) | Throughput ( $\sigma$ /s) |
|-------------------|-----|------------------------|-------------------|-------------------|---------------------------|
| <i>Ed25519</i>    | X   | X                      | 65                | 2.8               | 14,749                    |
| <i>ML-DSA</i>     | ✓   | X                      | 262               | 3.8               | 3,762                     |
| <i>FN-DSA</i>     | ✓   | X                      | 157               | 1.3               | 6,317                     |
| <i>OO-Ed25519</i> | X   | 60                     | 2                 | 2.8               | 208,334                   |
| <i>OO-ML-DSA</i>  | ✓   | 27 <sup>†</sup>        | 65                | 3.8               | 14,534                    |
| <i>OO-FN-DSA</i>  | ✓   | 42 <sup>†</sup>        | 22                | 1.3               | 42,918                    |
| <b>SOKO</b>       | ✓   | <b>0.5<sup>†</sup></b> | <b>1</b>          | <b>1.3</b>        | <b>433,537</b>            |

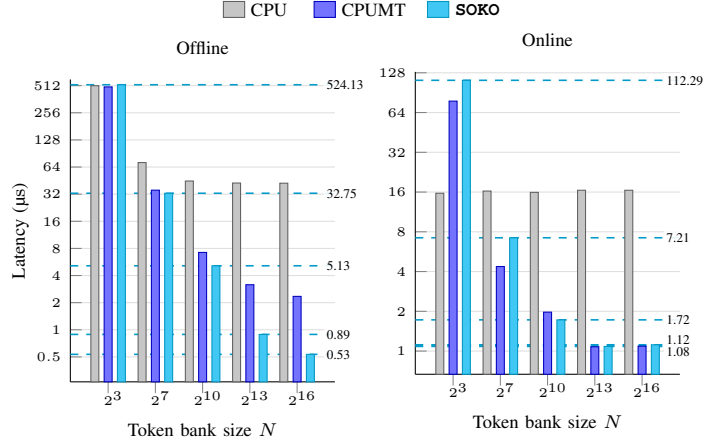


Fig. 3: Offline per-signature latency.

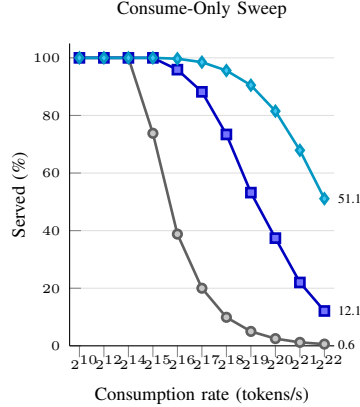


Fig. 5: Consume-only served percentage under rising token-consumption demand.

Fig. 4: Online per-signature latency.

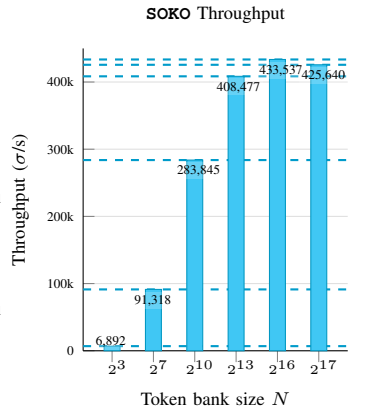


Fig. 6: **SOKO** end-to-end throughput.

#### B. Performance Evaluation

Table I and Fig. 3–6 characterize preprocessing, online signing, verification, and sustained token-generation throughput. The online/offline transformation moves message-independent work out of the latency-critical path. This is visible in Table I: standard FN-DSA requires 157  $\mu$ s per signature online, whereas OO-FN-DSA reduces the online phase to 22  $\mu$ s; similarly, OO-ML-DSA reduces the online phase to 65  $\mu$ s. Verification costs remain in the same low range for the corresponding base schemes, so the dominant

improvement comes from shortening the sender hot path rather than from changes in verification. The throughput numbers reflect this shift directly, increasing from 6,317  $\sigma/s$  for FN-DSA to 42,918  $\sigma/s$  for OO-FN-DSA.

In Fig. 3 and 4, we isolate the token-bank behavior of OO-FN-DSA under three refill strategies: CPU, CPUMT, and **SOKO**. The offline plot reports amortized PreSign latency per token as the batch size increases, while the online plot reports the remaining per-signature completion latency once precomputed material is available. The online phase is already small for all optimized variants, staying near the low-single-digit microsecond range for the larger banks. In contrast, the offline phase shows the main scalability difference. As the bank size grows, **SOKO** reduces offline per-token latency from 524.13  $\mu s$  at  $N = 2^3$  to 0.53  $\mu s$  at  $N = 2^{16}$ , substantially below both CPU and CPUMT at the same batch sizes. This reduction is reflected in Figure 6, where end-to-end throughput increases with bank size and reaches 433,537  $\sigma/s$  at  $N = 65,536$ .

Before interpreting Fig. 5, note that this experiment removes the online signing step and directly consumes precomputed tokens. The purpose is to isolate refill capacity, rather than end-to-end signer throughput, under sustained demand. All three systems begin with a filled queue and use threshold-triggered refills while the token-consumption rate is increased from  $2^{10}$  to  $2^{22}$  tokens/s. Under this setup, CPU sustains 100% service only up to 16,384 tokens/s and then degrades rapidly, CPUMT sustains full service through 32,768 tokens/s and declines more gradually, **SOKO** remains near 100% across a much wider range and still serves 51.1% of demand at 4,194,304 tokens/s. The sweep therefore shows that once the online phase is removed, the dominant performance difference comes from the rate at which each system can regenerate tokens.

This interpretation is consistent with the larger-batch refill measurements. With a 65,536-token refill batch, CPU reaches 20,057 refilled tokens/s, CPUMT reaches 435,215 tokens/s, and **SOKO** reaches 2,861,267 tokens/s. Thus, at the same refill granularity, **SOKO** provides a substantially higher PreSign generation rate than either CPU baseline.

## V. CONCLUSION

High-throughput and time-critical communication systems require authentication mechanisms that minimize signing latency, verification cost, and sender-side throughput bottlenecks. This paper introduced **SOKO**, a GPU-accelerated online/offline post-quantum signature framework based on OO-FN-DSA. **SOKO** moves message-independent token generation to an asynchronous GPU preprocessing phase, while keeping latency-sensitive online signing on the CPU. A threshold-based ring-buffer queue decouples GPU token production from CPU token consumption, allowing the framework to sustain high signing rates without placing GPU launch, synchronization, or transfer overhead on the real-time signing path.

Our evaluation shows that **SOKO** achieves 433,537 signatures/s with 2.3  $\mu s$  cryptographic end-to-end delay, corresponding to a  $115\times$  improvement over ML-DSA and a  $69\times$  improvement over standard FN-DSA. These results demonstrate that GPU-accelerated offline preprocessing can make online/offline FN-DSA practical for high-rate post-quantum broadcast authentication. Overall, **SOKO** provides a system-level path toward deployable PQ authentication in time-critical networks where conventional PQ signing alone is insufficient to meet throughput and latency requirements.

## REFERENCES

- [1] ETSI, "Intelligent Transport Systems (ITS); Security; Security header and certificate formats; Release 2," ETSI TS 103 097 V2.1.1, Oct. 2021. [Online]. Available: [https://www.etsi.org/deliver/etsi\\_ts/103000\\_103099/103097/02.01.01\\_60/ts\\_103097v020101p.pdf](https://www.etsi.org/deliver/etsi_ts/103000_103099/103097/02.01.01_60/ts_103097v020101p.pdf)
- [2] 3GPP, "NR; NR and NG-RAN overall description," 3GPP TS 38.300, Release 17, 2023.
- [3] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proc. 35th Annual Symposium on Foundations of Computer Science*, 1994, pp. 124–134.
- [4] National Institute of Standards and Technology, "FIPS 204: Module-Lattice-Based Digital Signature Standard," Aug. 2024. [Online]. Available: <https://csrc.nist.gov/pubs/fips/204/final>
- [5] National Institute of Standards and Technology, "FIPS 205: Stateless Hash-Based Digital Signature Standard," Aug. 2024. [Online]. Available: <https://csrc.nist.gov/pubs/fips/205/final>
- [6] IETF COSE Working Group, "FN-DSA for JOSE and COSE," Internet-Draft draft-ietf-cose-falcon, 2025. [Online]. Available: <https://datatracker.ietf.org/doc/draft-ietf-cose-falcon/>
- [7] U.S. Congress, "Quantum Computing Cybersecurity Preparedness Act," Public Law 117-260, Dec. 2022.
- [8] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang, "Falcon: Fast-Fourier lattice-based compact signatures over NTRU," NIST PQC Round 3 Submission, 2020.
- [9] D. Sikeridis, P. Kampanakis, and M. Devetsikiotis, "Post-Quantum Authentication in TLS 1.3: A Performance Study," in *Proc. Network and Distributed System Security Symposium (NDSS)*, 2020.
- [10] S. E. Nouma and A. A. Yavuz, "Lightweight and Resilient Signatures for Cloud-Assisted Embedded IoT Systems," arXiv:2409.13937, 2024. [Online]. Available: <https://arxiv.org/abs/2409.13937>
- [11] S. Even, O. Goldreich, and S. Micali, "On-line/off-line digital signatures," in *Advances in Cryptology—CRYPTO '89*, Lecture Notes in Computer Science, vol. 435. Springer, 1990, pp. 263–275.
- [12] A. Shamir and Y. Tauman, "Improved online/offline signature schemes," in *Advances in Cryptology—CRYPTO 2001*, Lecture Notes in Computer Science, vol. 2139. Springer, 2001, pp. 355–367.
- [13] M. R. Albrecht, N. Gama, J. Howe, and A. K. Narayanan, "Post-Quantum Online/Offline Signatures," Cryptology ePrint Archive, Paper 2025/117, 2025.
- [14] Y. Zhang, R. Behnia, A. A. Yavuz, R. Ebrahimi, and E. Bertino, "Efficient Full-Stack Private Federated Deep Learning With Post-Quantum Security," *IEEE Transactions on Dependable and Secure Computing*, vol. 22, no. 5, pp. 5567–5583, 2025.
- [15] S. C. Seo, S. W. An, and D. Choi, "Accelerating Falcon Post-Quantum Digital Signature Algorithm on Graphic Processing Units," *Computers, Materials & Continua*, vol. 75, no. 1, pp. 1711–1731, 2023.
- [16] R. Dai, J. Dong, M. Qiu, Z. Dong, F. Xiao, and J. Lin, "GOLF: Unleashing GPU-Driven Acceleration for FALCON Post-Quantum Cryptography," Cryptology ePrint Archive, Paper 2025/749, 2025.
- [17] W.-K. Lee, R. K. Zhao, R. Steinfeld, A. Sakzad, and S. O. Hwang, "High Throughput Lattice-Based Signatures on GPUs: Comparing Falcon and Mitaka," *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 4, pp. 675–692, 2024.
- [18] E. Barker and J. Kelsey, "Recommendation for Random Number Generation Using Deterministic Random Bit Generators," NIST Special Publication 800-90A Revision 1, June 2015. [Online]. Available: <https://doi.org/10.6028/NIST.SP.800-90Ar1>