

GPU-Assisted FAEST v3 Signing: Challenge Grinding and CPU–GPU Pipelining

Ha-Gyeong Kim¹[0009–0008–5840–5517], Si-Woo Eum¹[0000–0002–9583–5427],
Seung-Won Lee¹[0009–0009–9583–1387], Min-Ho Song¹[0009–0007–6277–0069], and
Hwa-Jeong Seo¹[0000–0003–0069–9061]

Hansung University, Seoul, Korea
kimhaha4420@gmail.com, shuraatum@gmail.com, dkajdfhd1@gmail.com,
smino0906@gmail.com, hwajeong84@gmail.com

Abstract. Applying a GPU to FAEST v3 signing requires more than exploiting parallel computation: challenge grinding must preserve the minimum accepting counter selected by the original signing procedure, and single-sign latency and multi-sign throughput must be evaluated under distinct execution conditions. We implement GPU-assisted challenge grinding in which the GPU filters candidates over contiguous counter ranges in parallel, while the CPU performs the final acceptance checks in increasing counter order. For multiple independent signatures, we construct a CPU–GPU hybrid pipeline with two alternating signing-state slots, allowing CPU work for one slot to proceed while the GPU batch for the other slot is executing. Across the six non-EM FAEST v3 parameter sets evaluated on the same CPU–GPU system, the CPU–GPU path has lower representative single-sign latency for four parameter sets, whereas the optimized CPU path is lower for two. The largest representative same-session CPU/CPU–GPU latency ratio is 1.602799 for FAEST-192s. For multi-sign throughput with $B \in \{2, 4, 8\}$, the CPU–GPU path has higher representative throughput in 9 of the 18 parameter–batch-size combinations, while the CPU path is higher in the remaining 9 under the predefined primary aggregation rule. Correctness validation confirms that GPU-assisted challenge grinding produces the same minimum accepting counter and final signature as the reference path for the evaluated configurations. These results show that the performance effect of GPU assistance is not uniform across the evaluated FAEST v3 signing configurations and depends on both the parameter set and whether the execution targets single-sign latency or the throughput of multiple independent signatures.

Keywords: FAEST · Post-Quantum Signatures · GPU · CUDA · Challenge Grinding · Throughput

1 Introduction

FAEST is a post-quantum signature scheme that constructs zero-knowledge proofs from symmetric-key primitives and the VOLE-in-the-Head paradigm and converts them into digital signatures via the Fiat–Shamir transformation [2].

The non-EM parameter sets of FAEST v3 use AES-based one-way functions, and signing includes repeated computation together with challenge grinding to determine the final challenge [2].

Although GPUs can process many operations in parallel, simply moving the FAEST v3 signing path to a GPU does not guarantee the same result as the original signing procedure. In particular, challenge grinding repeatedly recomputes the final challenge for increasing counter values and checks not only the final-bit condition but also whether the batch all-but-one vector commitment (BAVC) opening determined by that challenge succeeds [2]. Because the signer uses the first counter that satisfies all of these conditions, parallel evaluation of multiple counters on a GPU cannot simply accept whichever candidate finishes first. An execution structure is therefore required that exploits GPU parallelism while preserving the minimum accepting counter selected by the original FAEST v3 signing procedure.

Reducing the latency of one signature and processing multiple independent signatures together also impose different execution conditions. For a single signature, the relevant question is how GPU use changes the latency of the complete signing call. When multiple requests are available concurrently, however, independence across requests can be used to overlap CPU and GPU work. GPU assistance should therefore be evaluated separately for single-sign latency and for the throughput of multiple independent signatures rather than summarized by a single performance number.

We design CPU-GPU hybrid execution for the FAEST v3 signing path while distinguishing these two execution conditions. For a single signature, the GPU filters candidates satisfying the final-bit condition over contiguous counter ranges in parallel, while the CPU rechecks the candidates in increasing counter order and performs the final-challenge and BAVC-opening decisions. This structure evaluates multiple counters concurrently on the GPU while preserving the minimum-accepting-counter semantics required by the original signing procedure. The final implementation applies this GPU-assisted grinding path to the single-sign paths of FAEST-128s and FAEST-192s.

For multiple independent signatures, we process B requests as one workload unit and alternate between two signing-state slots so that GPU batch execution for one slot overlaps with CPU work for the other. Each request retains an independent state and signature, while the GPU side uses one shared execution resource so that at most one GPU batch is in flight at any time. This path uses the GPU pipeline for all six non-EM parameter sets while retaining challenge grinding on the existing CPU path.

Our evaluation focuses on three research questions. First, how can challenge candidates be filtered in parallel on a GPU while preserving the minimum-accepting-counter semantics of FAEST v3? Second, across the six non-EM parameter sets, how does the single-sign latency of the resulting full CPU-GPU hybrid path compare with that of an optimized CPU path on the same system? Third, when multiple independent signatures are processed together, how does the throughput of the CPU-GPU pipeline vary with the parameter set and batch

size B relative to an optimized CPU path processing the same number of requests concurrently?

To answer these questions, we measure all six non-EM FAEST v3 parameter sets on the same CPU–GPU system. For single signatures, we compare the optimized CPU full-sign path with the final CPU–GPU hybrid full-sign path; for multiple independent signatures, we compare the CPU–GPU hybrid pipeline with a concurrent CPU path processing the same B requests for $B \in \{2, 4, 8\}$. Each execution path is validated for correctness before performance measurement, and the single-sign and throughput evaluations use distinct measurement boundaries and aggregation procedures.

For single-sign latency, the CPU–GPU hybrid path has lower representative latency for four of the six parameter sets, while the CPU baseline is lower for the other two. The largest representative comparison ratio is 1.602799 for FAEST-192s, for which the representative latency of the CPU–GPU hybrid path is 37.54% lower. Thus, no single-sign performance improvement common to all evaluated parameter sets is observed.

The throughput results for multiple independent signatures likewise do not point in a single direction. Under the predefined primary aggregation rule, the CPU–GPU hybrid path has higher representative throughput in 9 of the 18 conditions formed by six parameter sets and $B \in \{2, 4, 8\}$, while the CPU baseline is higher in the remaining 9. For some parameter sets, the relative direction of the two paths also changes with B . We therefore report the effect of GPU assistance separately by parameter set and execution condition rather than summarizing it with one universal speedup.

The main contributions of this work are as follows.

- We implement GPU-assisted challenge grinding that preserves the minimum-accepting-counter semantics of FAEST v3 by combining parallel GPU filtering of preliminary candidates with final CPU acceptance checks in increasing counter order.
- We construct a CPU–GPU pipeline for multiple independent FAEST signing requests using two signing-state slots and a shared GPU execution resource to overlap CPU processing with GPU batch execution.
- We evaluate single-sign latency and equal-batch-size throughput for all six non-EM FAEST v3 parameter sets by comparing the final CPU–GPU hybrid paths with optimized CPU baselines.

We do not claim pinned memory or slot alternation themselves as new general-purpose GPU techniques, nor do we attribute the observed performance differences to the independent effect of any single component or propose a general runtime policy that automatically selects between CPU and GPU paths.

The remainder of this paper is organized as follows. Sections 2.1 and 2.2 summarize the FAEST v3 signing procedure and related work, and Section 3.1 onward presents GPU-assisted challenge grinding and the CPU–GPU pipeline for multiple independent signatures. Section 4.1 onward defines the experimental environment, measurement protocol, and validation methodology, and Section 5

reports the performance results. Finally, Section 6 discusses the interpretation and limitations of the results, and Section 7 concludes the paper.

2 Background and Related Work

2.1 FAEST v3 Signing and Challenge Grinding

FAEST is a post-quantum signature scheme based on symmetric-key cryptographic components; the six non-EM parameter sets evaluated in this work construct signatures by proving in zero knowledge that the signer knows the secret key of an AES-based one-way function [4,2,9]. FAEST combines the VOLE-in-the-Head paradigm with QuickSilver-style proof techniques and applies the Fiat–Shamir transformation to replace the verifier challenges of an interactive proof with hash-derived challenges [4,13,6]. Rather than restating the full proof system, this section focuses on final-challenge generation, challenge grinding, and BAVC opening, which are the parts directly connected to the GPU design developed later.

The signer first generates the VOLE commitment and correction data and derives the first challenge chall_1 from the message hash and commitment-related transcript [2]. It then computes the commitment to the extended witness and the values required for the VOLE consistency check, derives the second challenge chall_2 , and uses it to compute the QuickSilver proof values $\tilde{a}_0, \dots, \tilde{a}_{d_{zk}-1}$. The final Fiat–Shamir challenge chall_3 is generated by hashing these proof values together with a 32-bit counter, and both opening rejection and challenge grinding are applied at this stage [2]. All d_{zk} proof values are used to generate chall_3 , but the final signature stores only $\tilde{a}_1, \dots, \tilde{a}_{d_{zk}-1}$, while the verifier recomputes $\tilde{a}_0$ [2].

FAEST v3 uses a batch all-but-one vector commitment (BAVC) for its VOLE commitment [2,3]. BAVC excludes one position from the opening of each of several vectors and enables the verifier to reconstruct the values corresponding to the remaining positions [3]. The first $\lambda - w_{\text{grind}}$ bits of the final challenge are partitioned as

$$I = (\Delta_0, \dots, \Delta_{\tau-1})$$

where each Δ_i determines the position left unopened in the i th BAVC vector [2]. To enable reconstruction of the remaining leaves, the BAVC opening reveals the necessary internal seeds of the GGM tree while excluding these positions [7,3,2]. If opening the selected positions would require more than T_{open} revealed λ -bit seeds, the opening returns $\perp$ and the corresponding challenge is rejected [2]. Thus, final challenge acceptance depends not only on a hash condition but also on the BAVC opening determined by that challenge.

Challenge grinding additionally requires the final w_{grind} bits of the final challenge to be zero [2]. The signer increments the counter, recomputes chall_3 , and attempts the BAVC opening only for challenges satisfying this condition [2]. Satisfying the final-bit condition alone does not select the counter; the BAVC opening determined by the challenge must also succeed for the counter to be accepted.

Following the specification notation, the final challenge for counter c is written as [2]:

$$\chi(c) = H_3^2(\text{chall}_2 \parallel \text{ToBits}((\tilde{a}_j)_{j=0}^{d_{zk}-1}; \lambda, d_{zk}) \parallel \text{BitDec}(c, 32); \lambda). \quad (1)$$

Here, $\text{BitDec}(c, 32)$ is the specification-level transformation that represents the counter as a 32-bit little-endian bit string [2]. Its byte-level handling in the GPU implementation is described in Section 3.2.

We first define whether the final w_{grind} bits are all zero as

$$Z(c) = [\chi(c)[\lambda - w_{\text{grind}} .. \lambda] = 0^{w_{\text{grind}}}]. \quad (2)$$

However, $Z(c) = 1$ does not mean that counter c is finally accepted. A challenge passing this predicate is transformed by the specification’s `DecodeAllChall3` procedure into the positions excluded from the BAVC opening, and the counter is accepted only if the opening for those positions also succeeds [2]. We therefore define final acceptance of counter c as

$$A(c) = \begin{cases} Z(c) = 1 \wedge \\ 1, & \text{BAVC. Open}(\text{decom}, \text{DecodeAllChall3}(\chi(c); \text{param}); \\ & \text{param}, \text{param}_{\text{VOLE}}) \neq \perp, \\ 0, & \text{otherwise.} \end{cases} \quad (3)$$

The FAEST v3 signing procedure examines counters sequentially from $c = 0$ and terminates at the first counter satisfying $A(c) = 1$ [2]. Accordingly, the counter selected by the specification can be expressed as the minimum accepting counter

$$c^* = \min \{c \in \{0, \dots, 2^{32} - 1\} \mid A(c) = 1\}. \quad (4)$$

If this set is empty, signing fails.

Table 1 summarizes the main specification values of the six non-EM FAEST v3 parameter sets evaluated in this work [2]. Here, λ is the security parameter, ℓ is the witness length in bits used by the zero-knowledge proof, and τ is the number of small VOLE instances. The parameter w_{grind} is the number of suffix bits required to be zero in the final challenge, and T_{open} is the upper bound on the number of revealed λ -bit seeds in the GGM-tree opening. For all six evaluated parameter sets, the number of λ -bit blocks in each leaf commitment is $n_{\text{leafcom}} = 3$, and the maximum degree of the QuickSilver constraints is $d_{zk} = 7$ [2].

This sequential selection rule must also be preserved under GPU parallelization. Even if the final-bit condition is evaluated simultaneously for many counters, selecting whichever candidate is first discovered or first completed on the GPU does not guarantee the same c^* as the original signing procedure. Section 3.2 describes a design that preserves the minimum accepting counter by combining parallel GPU candidate filtering with final CPU acceptance checks in increasing counter order.

Table 1. Main specification values of the evaluated non-EM FAEST v3 parameter sets

Parameter	λ	ℓ (bits)	τ	w_{grind}	T_{open}	Signature size (bytes)
FAEST-128s	128	960	11	7	102	4,066
FAEST-128f	128	960	17	8	108	5,170
FAEST-192s	192	1728	16	12	162	9,410
FAEST-192f	192	1728	24	8	163	11,738
FAEST-256s	256	2208	22	6	225	16,626
FAEST-256f	256	2208	33	8	229	20,856

2.2 Related Work and Research Positioning

VOLE-in-the-Head provides the direct cryptographic foundation of FAEST. Baum et al. introduced VOLE-in-the-Head, which transforms VOLE-based zero-knowledge protocols in the designated-verifier setting into publicly verifiable public-coin protocols, and applied the technique to FAEST, a post-quantum signature proving knowledge of an AES secret key [4]. Subsequently, *One Tree to Rule Them All* improved the all-but-one vector commitment (BAVC) used by VOLE-in-the-Head and MPC-in-the-Head schemes and introduced a small proof-of-work mechanism for parameter tuning [3]. *Shorter, Tighter, FAESTer* further improved FAEST efficiency and its QROM analysis through, among other changes, an optimization that replaces hash calls in the underlying vector commitment with AES-based pseudorandom-generator calls [1]. Our work does not reuse performance numbers from these studies; instead, it directly implements and evaluates the non-EM signing path of the current FAEST v3 specification [2].

In addition to the reference implementation, the official FAEST software provides architecture-specific optimized implementations for x86-64 and AArch64, including the use of instruction sets such as AVX2 and AES-NI on the x86-64 path [5]. Our CPU baseline is likewise an optimized full-sign path rather than the reference implementation, and all CPU measurements are obtained directly from the research implementation on the same evaluation system described in Section 4.1, rather than copied from public benchmark tables.

GPU implementations have also been studied for other post-quantum signature schemes. Shen et al. presented a high-throughput GPU implementation of Dilithium combining batching, memory pools, dynamic work scheduling, asynchronous execution, and multiple CUDA streams [12]. Ning et al. proposed GRASP, which executes parallelizable operations in SPHINCS+ signing and verification on a GPU [10]. Because Dilithium and SPHINCS+ have computational structures and signing procedures different from FAEST, we do not directly compare their performance numbers with ours; instead, we cite them as examples of batching independent work and exploiting GPU parallelism in other post-quantum signatures.

Against this background, we focus on GPU-assisted challenge grinding that preserves the minimum-accepting-counter semantics of FAEST v3 and on CPU-

GPU hybrid execution for multiple independent signatures. For the six non-EM parameter sets, we compare single-sign latency and throughput at $B \in \{2, 4, 8\}$ against optimized CPU baselines on the same system.

3 Design and Implementation

3.1 Design Goals and Execution Policy

Our design goal is to exploit GPU parallelism without changing the signing semantics of FAEST v3. Because challenge grinding in Section 2.1 selects the minimum accepting counter c^* after including the BAVC-opening result, we assign preliminary candidate filtering to the GPU and retain the order-sensitive final acceptance decision on the CPU.

Single-sign and multi-sign workloads have different performance objectives and expose different forms of parallelism. Throughout this paper, B denotes the number of mutually independent FAEST signing requests processed together; it is unrelated to the BAVC parameter τ , and each request produces a separate FAEST signature. For $B = 1$, we evaluate the single-sign latency of the complete signing call, whereas for $B > 1$ we evaluate effective throughput obtained from parallelism across independent requests and CPU–GPU overlap; consequently, the two regimes use different execution paths.

The optimized FAEST CPU implementation uses XKCP to process four Keccak permutations in parallel at the instruction level during challenge-grinding hash computation [2]. We refer to this CPU grinding path as *4-way CPU grinding*.

For $B = 1$, GPU-assisted grinding partitions consecutive counters into chunks of size C . The final implementation uses GPU-assisted grinding with $C = 4096$ for FAEST-128s and $C = 32768$ for FAEST-192s, while the remaining four parameter sets retain 4-way CPU grinding.

For $B > 1$, all six parameter sets use the GPU pipeline for multiple independent signatures, while challenge grinding remains on the 4-way CPU path. Thus, this path is a CPU–GPU hybrid execution that combines CPU grinding, GPU batch work, and the remaining signing stages; throughput is evaluated for $B \in \{2, 4, 8\}$.

Table 2 summarizes the parameter-specific execution policy fixed before performance evaluation; the CPU baseline processing the same B requests is defined in Section 4.2.

The parameter sets and chunk sizes used for GPU-assisted grinding were chosen before the formal performance campaign using development-stage cost analysis and a limited set of candidate-chunk measurements. The cost analysis selected FAEST-192s as the primary target and FAEST-128s as the secondary target based on the share of grinding-hash cost that could be removed. We tested $C \in \{4096, 8192, 16384\}$ for FAEST-128s and $C \in \{16384, 32768, 65536\}$ for FAEST-192s. A candidate was considered eligible only if the median of the GPU replacement path was lower than that of the 4-way CPU hash-search path in both development measurement blocks; among eligible candidates, we selected

Table 2. Parameter-specific execution paths fixed before performance evaluation

Parameter	$B = 1$ grinding	Chunk size C	$B > 1$ evaluation path	$B > 1$ grinding
FAEST-128s	GPU-assisted	4096	Independent-signature GPU pipeline	4-way CPU
FAEST-128f	4-way CPU	–	Independent-signature GPU pipeline	4-way CPU
FAEST-192s	GPU-assisted	32768	Independent-signature GPU pipeline	4-way CPU
FAEST-192f	4-way CPU	–	Independent-signature GPU pipeline	4-way CPU
FAEST-256s	4-way CPU	–	Independent-signature GPU pipeline	4-way CPU
FAEST-256f	4-way CPU	–	Independent-signature GPU pipeline	4-way CPU

$C = 4096$ and $C = 32768$, respectively, because they had the lowest pooled GPU median. After screening the final integrated implementation, the selected configurations were frozen before the formal measurement protocol and campaign began, and neither the target parameter sets nor the chunk sizes were reselected based on the formal performance results. This selection applies only to the tested development candidates and does not imply that the same chunk sizes are optimal on other systems or over untested chunk sizes.

3.2 GPU-Assisted Challenge Grinding

Preserving c^* as defined in Section 2.1 requires GPU parallelization not to change the counter-selection order. We therefore divide grinding into preliminary candidate filtering on the GPU and final acceptance checks on the CPU in increasing counter order. For each contiguous counter chunk, the GPU evaluates the predicate in Eq. (2) in parallel and marks only positions whose final w_{grind} bits are all zero in a bitmap. The CPU scans this bitmap from lower to higher counters and recomputes the final challenge and performs the BAVC opening only for marked candidates. A GPU hit is therefore only a preliminary candidate requiring an additional CPU decision, not the final selected counter.

The portion of the final-challenge hash input preceding the counter remains constant throughout one signing operation. At the specification level, we write

this fixed prefix as

$$P = \text{chall}_2 \parallel \text{ToBits}\left((\tilde{a}_j)_{j=0}^{d_{zk}-1}; \lambda, d_{zk}\right). \quad (5)$$

The implementation packs the three input regions required for the second challenge and the QuickSilver check and proof directly into a pinned transfer buffer; the fixed input is 168 bytes for FAEST-128s and 248 bytes for FAEST-192s. This input is transferred to the GPU once per grinding invocation, preabsorbed into a Keccak state, and then reused for every counter chunk in that invocation.

For each counter, its four-byte little-endian serialization is appended to the fixed input, followed by the domain-separation byte `0x0b` corresponding to H_3^2 and then the SHAKE padding. This counter representation uses the same bit ordering as $\text{BitDec}(c, 32)$ in the specification, and `0x0b` is the domain-separation value used by the FAEST definition of H_3^2 [2]. FAEST-128s uses SHAKE128 to generate a 16-byte challenge, whereas FAEST-192s uses SHAKE256 to generate a 24-byte challenge [2,8]. The GPU checks only the final-bit condition and records the result in the bitmap; it does not determine whether the BAVC opening succeeds.

The full counter space is partitioned into non-overlapping contiguous chunks of size C and processed from lower to higher counters; for starting counter b , the GPU evaluates exactly $[b, b + C)$. Because both final values $C = 4096$ and $C = 32768$ divide 2^{32} exactly, every GPU launch processes exactly C counters. The host stores b in a 64-bit integer and verifies that it remains within the 32-bit counter range before converting it to the GPU start value, preventing wraparound after the final chunk.

At the start of each chunk, the reused $C/8$ -byte GPU bitmap is cleared to zero so that no result from the previous chunk remains. The GPU then evaluates all C counters and constructs the device bitmap $\mathcal{B}_D$. After the completed bitmap is copied asynchronously to the host bitmap $\mathcal{B}_H$, the host waits for the existing CUDA stream to complete. Because operations submitted to the same CUDA stream execute in order, bitmap clearing, counter evaluation, and the device-to-host bitmap transfer remain ordered within a single stream [11]. Only after this sequence completes does the CPU scan the host bitmap from lower to higher counters.

When the CPU encounters a marked candidate, it recomputes $\chi(c)$ for that counter in the same manner as the signing path. It checks the final-bit condition on the recomputed challenge and uses the specification’s `DecodeAllChall3` procedure to derive the positions excluded from the BAVC opening. The existing BAVC opening procedure is then executed without modification. If the opening returns $\perp$, the counter is rejected and the CPU checks the next larger candidate in the same chunk. The implementation advances to the next contiguous chunk only after all candidates in the current chunk have been rejected.

Figure 1 illustrates the separation of roles and data movement between parallel GPU candidate filtering and final CPU acceptance in increasing counter order.

Algorithm 1 presents the GPU candidate-filtering and CPU final-acceptance procedure while omitting implementation details. The challenge interpretation and

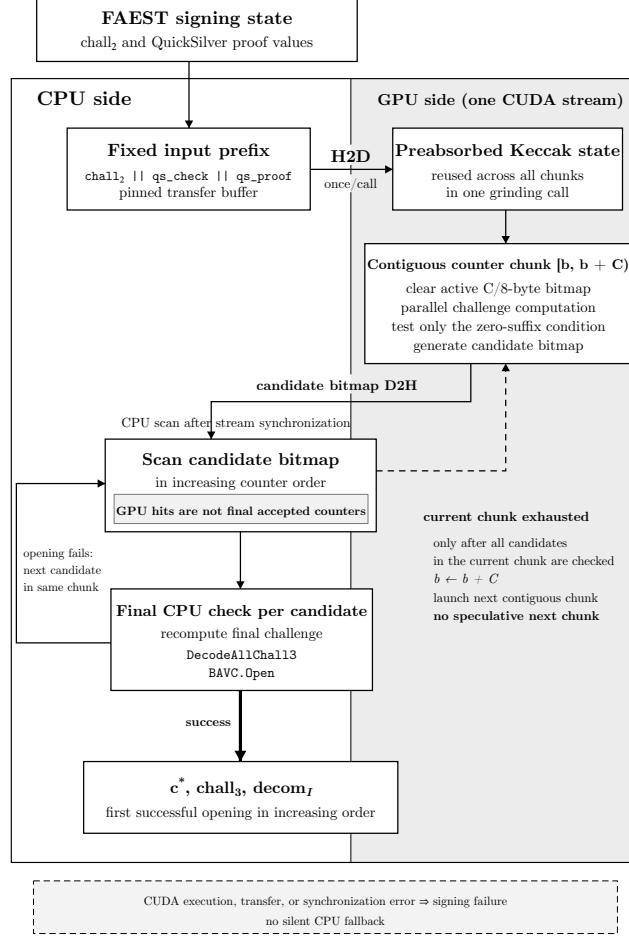


Fig. 1. CPU–GPU role separation in GPU-assisted challenge grinding

BAVC-opening notation in the algorithm refer to the corresponding procedures in the FAEST v3 specification.

Minimum-Counter Preservation. Assume that no execution, transfer, or synchronization error occurs and that the GPU marks $Z(c)$ from Eq. (2) exactly for every counter c . Also assume that the CPU recomputation of the final challenge, `DecodeAllChall3`, and `BAVC.Open` are identical to the reference signing procedure; that the counter space is traversed in contiguous chunks from lower to higher counters without omission or wraparound; and that candidates within each chunk are checked in increasing counter order. Under these conditions, Algorithm 1, which does not advance to the next chunk before all candidates in the current chunk have been decided, returns c^* from Eq. (4) together with its

Algorithm 1 Challenge grinding with GPU preliminary filtering and CPU ordered acceptance

Input: $\text{chall}_2, (\tilde{a}_j)_{j=0}^{d_{zk}-1}, \text{decom}$, chunk size C , public parameters

Output: $(c^*, \text{chall}_3, \text{decom}_I)$ or signing failure

```

1: Prepare fixed input  $P \leftarrow \text{chall}_2 \parallel \text{ToBits}((\tilde{a}_j); \lambda, d_{zk})$ 
2: Transfer  $P$  to the GPU and preabsorb it once
3: Set 64-bit host integer  $b \leftarrow 0$ 
4: while  $b + C \leq 2^{32}$  do
5:    $e \leftarrow b + C$ 
6:   Check that  $b$  is within the 32-bit counter range and convert it to the GPU start value
7:   Clear the reused GPU bitmap to zero
8:   GPU evaluates  $Z(c)$  for all  $c \in [b, e)$  and marks  $\mathcal{B}_D$ 
9:   Transfer  $\mathcal{B}_D$  to host bitmap  $\mathcal{B}_H$ 
10:  Wait for the existing CUDA stream to complete
11:  if GPU execution, transfer, or synchronization fails then
12:    return signing failure
13:  end if
14:  for  $o \leftarrow 0$  to  $C - 1$  do
15:    if  $\mathcal{B}_H[o] = 1$  then
16:       $c \leftarrow b + o$ 
17:      Recompute  $\chi(c)$  on the CPU
18:      if  $Z(c) = 1$  then
19:         $I \leftarrow \text{DecodeAllChall3}(\chi(c); \text{param})$ 
20:         $\text{decom}_I \leftarrow \text{BAVC.Open}(\text{decom}, I; \text{param}, \text{param}_{\text{VOLE}})$ 
21:        if  $\text{decom}_I \neq \perp$  then
22:          return  $(c, \chi(c), \text{decom}_I)$ 
23:        end if
24:      end if
25:    end if
26:  end for
27:   $b \leftarrow b + C$ 
28: end while
29: return signing failure

```

corresponding final challenge and BAVC-opening result whenever an accepting counter exists.

Indeed, every counter smaller than c^* either has $Z(c) = 0$ or, despite $Z(c) = 1$, fails the BAVC opening and is therefore rejected by the reference procedure as well. Because the chunks cover the counter space in increasing order and the CPU also scans each bitmap in increasing order, neither a candidate larger than c^* nor a candidate from a later chunk can be selected first. Therefore, the first candidate returned after a successful BAVC opening is exactly c^* . If no accepting counter exists, the implementation exhausts the full counter space and returns signing failure. If CUDA execution, transfer, or synchronization fails, this preservation conclusion does not apply and the implementation reports signing failure according to its error policy.

The pinned transfer/receive buffers and the device-side fixed input, preabsorbed Keccak state, and bitmap required for grinding are integrated into the internal signing context, allocated once, and reused across signatures rather than reallocating GPU memory for every signature. All grinding work uses the single CUDA stream already owned by the context to preserve ordering with preceding BAVC-related GPU work, and the next chunk is not launched speculatively before CPU processing of the current chunk finishes.

If GPU execution, host-GPU transfer, or waiting for CUDA-stream completion fails, the same signing request is not silently switched to CPU grinding; instead, signing fails. Exhausting the full counter space likewise returns signing failure without wraparound. After a signing operation completes, temporary data in the GPU and associated host buffers are cleaned up before reuse.

Section 4.3 describes the scope and results of implementation-level validation for the selected counter, final challenge, BAVC-opening result, and complete signature.

3.3 Multi-Sign GPU Pipeline

For $B > 1$, the execution path processes B mutually independent requests together, while each request retains its own input, internal state, counter, and final signature. The goal is to improve throughput by overlapping CPU and GPU work across independent signatures without changing the FAEST algorithm itself.

The final implementation alternates between two logical slots, each storing independent signing states for B requests, while both slots share one set of B CPU worker threads that process the stages before and after GPU execution. On the GPU side, the slots share one batch execution context, one persistent set of pinned transfer buffers, one device-buffer set, and one nonblocking CUDA stream; hence, at most one GPU batch is in flight at a time.

The CPU first prepares the B requests in the first slot and asynchronously submits GPU batch A; while GPU A executes, the shared CPU worker threads prepare the requests in the second slot. After preparation of the second slot completes, the implementation waits for the shared CUDA stream, applies A’s results to the first slot, and submits GPU batch B through the shared GPU batch context. While GPU B executes, the CPU continues the remaining signing stages for the first slot; repeating this schedule overlaps GPU execution for one slot with CPU work for the other.

Figure 2 schematically shows the initial batches and CPU-GPU overlap of the two slots; horizontal segment lengths are not proportional to measured execution time.

Within one GPU batch, three types of H2D transfer, the batched whole-leaf generation kernel (`whole_leaf`), and two types of D2H transfer are enqueued in order on the same CUDA stream, relying on the ordering semantics of a single stream [11]. The implementation neither executes two GPU batches concurrently nor overlaps different batches across multiple CUDA streams; pipeline overlap comes from CPU work on one slot while the GPU executes the other slot’s batch. When collecting results, the host waits for stream completion and applies the

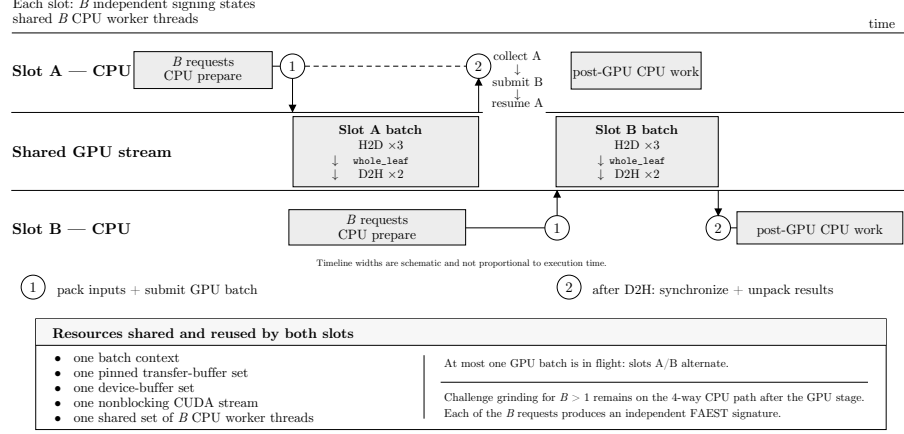


Fig. 2. Two-slot CPU–GPU pipeline for multiple independent signatures

output-buffer values to the corresponding request states; because CPU-side input packing and result copying remain, this is not a zero-copy path.

The pinned host buffers and GPU buffers used for GPU batches are allocated once for the lifetime of the shared batch context and reused across batches rather than repinned or reallocated for every batch [11]. Each request’s seed, IV, and hash IV are packed directly into the persistent pinned transfer buffers without an intermediate batch buffer in ordinary host memory.

For $B > 1$, challenge grinding remains on the 4-way CPU path for all six non-EM parameter sets and is not included in the GPU batch. Accordingly, the evaluated path is a CPU–GPU hybrid execution combining GPU batch processing, CPU grinding, and the remaining signing procedure.

This work combines pinned transfers and two-slot overlap for independent FAEST signing, and the performance evaluation targets the final pipeline as a whole rather than the individual components in isolation.

The final throughput evaluation compares this pipeline with an optimized CPU baseline processing the same B independent requests for the six non-EM parameter sets and $B \in \{2, 4, 8\}$; the measurement boundaries and aggregation rules are defined in Section 4.2.

4 Evaluation Methodology and Validation

4.1 Experimental Setup

All performance measurements were conducted on the same x86-64 system equipped with an Intel Core i7-13700KF CPU and an NVIDIA GeForce RTX 4080 16 GB GPU. The CPU baseline and CPU–GPU hybrid implementations were measured on the same hardware, and Table 3 summarizes the principal hardware and build environment of the final formal evaluation.

Table 3. Hardware and build environment for the final formal evaluation

Item	Configuration
CPU	Intel Core i7-13700KF
GPU	NVIDIA GeForce RTX 4080, 16 GB
GPU driver	535.309.01
CUDA Toolkit	12.0
NVCC	12.0.140
NVCC host compiler	<code>/usr/bin/g++-12</code>
Host C++ compiler	GCC/G++ 16.2.1 20260905
CPU compilation options	<code>-O3, -march=native, -mtune=native</code>
CUDA target and optimization	<code>sm_89, -O2</code>
Link-time optimization	Not used
Evaluated parameter sets	FAEST-128s, FAEST-128f, FAEST-192s, FAEST-192f, FAEST-256s, FAEST-256f
Fixed input corpus	Deterministic SHAKE256-based generation, 4 keys per parameter set, 32-byte messages, ρ length $\lambda/8$ bytes

The common CPU code of the CPU baseline and CPU–GPU hybrid paths was built with GCC/G++ 16.2.1 using the same major optimization settings, and both paths use identically built FAEST and XKCP libraries. The `/usr/bin/g++-12` entry in Table 3 is the host compiler used by NVCC when processing `.cu` sources; it does not mean that the general C++ signing code of the CPU–GPU hybrid path was separately built with GCC 12.

We evaluate the six non-EM FAEST v3 parameter sets, excluding FAEST-EM. For single signatures, we compare the optimized CPU path with the final CPU–GPU hybrid path defined in Section 3.1; for multiple independent signatures, we compare an optimized concurrent CPU path processing the same B requests with the CPU–GPU hybrid pipeline in Section 3.3. The two evaluations use different measurement boundaries, which are defined separately in Section 4.2.

All comparisons use the same fixed input corpus generated deterministically with SHAKE256 in advance. Each parameter set uses four keys and 32-byte messages, and the signing randomness ρ has length $\lambda/8$ bytes. Sample index i is assigned key $i \bmod 4$, and the same input-to-sample mapping is used for the CPU and CPU–GPU hybrid paths.

Before performance measurement, identifiers for the evaluated source, binaries, and input corpus were fixed; the measurement artifacts preserve the SHA-256 hashes of source and binaries, the input-corpus identifier, execution environment, actual invocation arguments, and raw measurements. These records are cross-referenced to verify the provenance of aggregated measurements, while measurement eligibility and execution-environment integrity checks are described in Section 4.4.

4.2 Measurement Boundaries and Aggregation

We distinguish single-sign latency, diagnostic measurements of internal execution intervals, and throughput for multiple independent signatures. The primary single-sign comparison uses M0 and M1, where M1 is the CPU–GPU hybrid path defined in Section 3.1; M2 and M3 are diagnostic measurements used to confirm internal measurement boundaries and execution relationships in the actual signing path. The throughput evaluation is conducted separately and is not treated as part of M0–M3.

Table 4 summarizes the boundary and role of each measurement, while Fig. 3 shows their execution relationships, including the separate M2 execution and the nested boundaries within M3. The box sizes and layout in the figure are schematic and are not proportional to actual execution time or relative duration.

For M0, the input and signing randomness ρ are already prepared before timing, and host time is measured from immediately before entry to immediately after return of the optimized CPU full-sign call. Signing work inside the call is included, whereas key and randomness generation, caller-side input preparation, signature verification, copying to an external output format, and measurement-side auxiliary processing are excluded.

For M1, a reusable GPU context already exists before timing, and host time is measured over the complete CPU–GPU hybrid signing call defined in Section 3.1, from immediately before entry to immediately after return. The boundary includes CPU and GPU computation, host–GPU transfers, and required synchronization, but excludes GPU-context creation and destruction, pre-call input preparation, signature verification, and copying to an external output format. Thus, M1 measures the latency of the defined complete hybrid signing call rather than an isolated GPU kernel.

M2 measures the actual `fc_run` interval, from entry to return, in a separate diagnostic execution of the signing path. The `fc_run` function invokes the GPU work that generates leaf keys and hashes and collects its results; GNU linker wrapping with `-Wl,-wrap=fc_run` is used to measure the actual call boundary. Because the operation is not rerun solely for timing, M2 observes the interval actually executed along the production signing path.

M3 records B1, B2, and B3 together within one and the same signing call. M3-B1 uses CUDA events to measure only execution of the whole-leaf-generation kernel (`whole_leaf`) on the existing CUDA stream and excludes the surrounding host–GPU transfers. M3-B2 measures the actual `fc_run` interval in the same call using host time, while M3-B3 measures the enclosing complete CPU–GPU hybrid signing call using host time. Diagnostic result access and the CUDA-event elapsed-time query occur only after both `fc_run` and the complete signing call return, and are therefore outside M3-B2 and M3-B3.

B1, B2, and B3 in M3 observe different execution boundaries and are not summed, and M2 from the separate diagnostic execution is likewise not added to M3-B1 to form a composite metric. M3-B3 is not a primary performance value replacing M1; it is used to observe the outer call boundary of the M3 diagnostic execution.

Table 4. Boundaries and roles of latency, diagnostic, and throughput measurements

ID	Measured target	Boundary and method	Role
M0	Optimized CPU full-sign path	Host time from immediately before to immediately after the core signing call with prepared input and ρ	CPU baseline for single-sign latency
M1	CPU–GPU hybrid full-sign path	Host time from immediately before to immediately after the complete hybrid signing call with a reusable GPU context already available	Primary single-sign comparison path
M2	Actual <code>fc_run</code> interval invoked during signing	Host time from entry to return of the actual <code>fc_run</code> call	Internal-boundary diagnostic
M3-B1	Whole-leaf-generation kernel within the same signing call	CUDA-event timing of only the <code>whole_leaf</code> kernel on the existing CUDA stream	GPU-kernel-interval diagnostic
M3-B2	<code>fc_run</code> interval within the same signing call	Host time from entry to return of the actual <code>fc_run</code> in the same signing call as M3	Boundary-consistency check against M2
M3-B3	Complete CPU–GPU hybrid signing interval of the M3 diagnostic execution	Host time of the outer full-sign interval of the same call	Outer call boundary containing B1 and B2; not used for interval summation
Throughput	CPU or CPU–GPU path processing B independent signatures	From the point at which requests can enter the first preparation stage until all signatures in the measured interval complete	Effective-throughput comparison for multiple independent signatures

Host intervals use `CLOCK_MONOTONIC_RAW`, whereas M3-B1 uses CUDA events because it targets a GPU-kernel interval; M3-B1 is therefore not interpreted as a host-clock measurement.

Each single-sign measurement block performs 100 warm-up iterations followed by 1000 measured iterations. Primary latency measurements repeat M0 and M1 in sessions S1, S2, and S3; the parameter order and the M0/M1 execution order are fixed in advance and rotated across the three sessions. Diagnostic measurements are conducted in separate sessions DG1, DG2, and DG3, rotating the parameter order and executing M2 before M3 for each parameter. Even when the indices

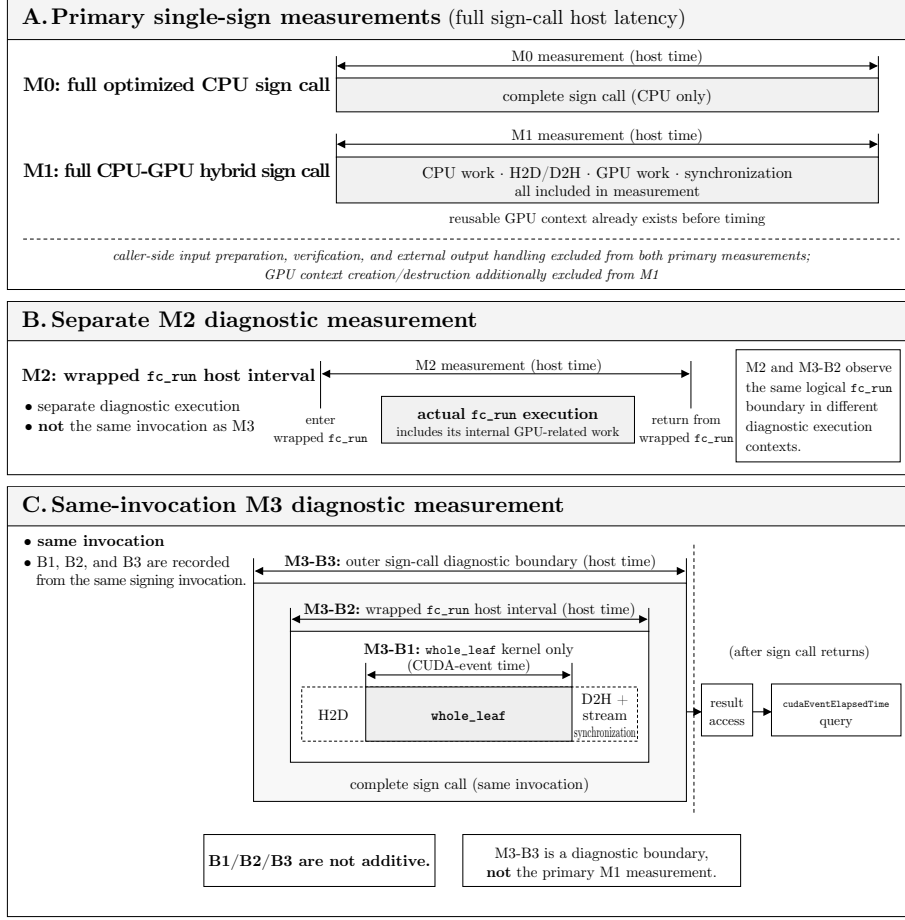


Fig. 3. Measurement boundaries and execution relationships for M0–M3

correspond, such as S1 and DG1, the sessions are not temporally paired, and we do not assume the sessions to be statistically independent samples.

For the primary single-sign M0/M1 measurements, the measured thread is pinned to logical CPU 0. This pinning applies to the measured thread only and does not imply that the CUDA runtime or all other helper threads are pinned to the same logical CPU.

Let p denote the parameter set, $s \in \{S1, S2, S3\}$ the primary measurement session, and $j \in \{0, 1\}$ the measurement path. For the 1000 measurements $t_{j,p,s,r}$ recorded in one block, the representative latency of each session is defined as

$$m_{j,p,s} = \text{median}_{1 \leq r \leq 1000} t_{j,p,s,r}. \quad (6)$$

For parameter set p and path j , the final representative latency is the median of the three session medians:

$$L_{j,p} = \text{median}_{s \in \{S1, S2, S3\}} m_{j,p,s}. \quad (7)$$

For the representative CPU/CPU-GPU comparison ratio, the M0/M1 median ratio is first computed within each session and the median of the three ratios is then taken:

$$R_p = \text{median}_{s \in \{S1, S2, S3\}} \left(\frac{m_{0,p,s}}{m_{1,p,s}} \right). \quad (8)$$

In general, R_p is therefore not equal to the simple ratio $L_{0,p}/L_{1,p}$ of the final representative latencies. Section 5.1 uses R_p from Eq. (8) as the primary ratio for the single-sign comparison. A value $R_p > 1$ indicates the direction in which M1 has lower representative latency, while $R_p < 1$ indicates the direction in which M0 is lower.

For M2 and M3, we likewise compute the median of the 1000 measurements within each DG session and then use the median of the three session medians as the representative value; M3-B1, M3-B2, and M3-B3 are aggregated separately. Section 4.4 discusses interpretation of these diagnostic values and the comparison between M2 and M3-B2.

The predefined measurement protocol does not specify thresholds for statistical significance, minimum effect size, or performance equivalence. Accordingly, subsequent comparisons are descriptive comparisons of the representative values and ratios defined above and do not constitute claims of statistical significance or performance equivalence between the paths.

The throughput evaluation is conducted separately from single-sign latency, and both the CPU and CPU-GPU hybrid paths process the same $B \in \{2, 4, 8\}$ independent requests concurrently. The CPU-GPU hybrid path uses the pipeline in Section 3.3; the difference in time-measurement units and block-level aggregation between the two paths is part of the protocol fixed in advance.

For the CPU baseline, each B -request batch is timed from the point at which the requests are released for execution until all B signatures complete, whereas the CPU-GPU hybrid implementation records the elapsed time of the complete pipeline interval that processes 200 consecutive B -request batches. For each condition, 20 warm-up batches are followed by 200 measured batches, and the same condition is repeated in two measurement blocks. Thus, the two paths process the same number of independent requests, but their time-measurement and block-aggregation units are not identical.

The CPU worker threads in the throughput evaluation are not explicitly pinned to specific logical CPUs, and placement between P-cores and E-cores is not separately controlled. The throughput results therefore reflect executions in which the operating-system scheduler places worker threads on the available CPU resources; actual per-worker core placement and migration histories are not controlled.

The CPU baseline and CPU-GPU hybrid implementation compute block-level representative throughput differently according to the predefined protocol. Let

p denote the parameter set, B the batch size, and $k \in \{1, 2\}$ the measurement block. All time variables in the following equations are expressed in seconds.

For the CPU baseline, if $t_{p,B,k,i}^{\text{CPU}}$ is the elapsed time of measured batch i , its throughput is

$$q_{p,B,k,i}^{\text{CPU}} = \frac{B}{t_{p,B,k,i}^{\text{CPU}}}. \quad (9)$$

The CPU block representative is the median of the 200 per-batch throughput values:

$$q_{\text{CPU},p,B,k} = \text{median}_{1 \leq i \leq 200} q_{p,B,k,i}^{\text{CPU}}. \quad (10)$$

For the CPU–GPU hybrid implementation, one measurement block processes 200 batches of B requests. If the elapsed time of the complete interval is $T_{p,B,k}^{\text{CPU-GPU}}$, the block representative throughput is

$$q_{\text{CPU-GPU},p,B,k} = \frac{200B}{T_{p,B,k}^{\text{CPU-GPU}}}. \quad (11)$$

For execution path $v \in \{\text{CPU}, \text{CPU-GPU}\}$, the final representative throughput is the median of the two block representatives from that path:

$$Q_{v,p,B} = \text{median}_{k \in \{1,2\}} q_{v,p,B,k}. \quad (12)$$

The representative throughput ratio between the two paths is

$$G_{p,B} = \frac{Q_{\text{CPU-GPU},p,B}}{Q_{\text{CPU},p,B}}. \quad (13)$$

A value $G_{p,B} > 1$ indicates the direction in which the CPU–GPU hybrid path has higher throughput under the predefined representative-throughput rule, whereas $G_{p,B} < 1$ indicates the direction in which the CPU baseline is higher. This ratio is not a statistical superiority or performance-equivalence decision.

Post-hoc Aggregation Sensitivity. After the formal measurements were complete, we performed an auxiliary analysis to examine the sensitivity of the results to the CPU baseline’s block-aggregation rule. This analysis does not modify the predefined primary measurement protocol or the primary results and uses all 200 batch elapsed times that had already passed eligibility checks in the formal CPU measurements. The auxiliary aggregate throughput for CPU block k is defined as

$$q_{\text{CPU},p,B,k}^{\text{agg}} = \frac{200B}{\sum_{i=1}^{200} t_{p,B,k,i}^{\text{CPU}}}, \quad Q_{\text{CPU},p,B}^{\text{agg}} = \text{median}_{k \in \{1,2\}} q_{\text{CPU},p,B,k}^{\text{agg}}. \quad (14)$$

The auxiliary throughput ratio is $G_{p,B}^{\text{agg}} = Q_{\text{CPU-GPU},p,B} / Q_{\text{CPU},p,B}^{\text{agg}}$. Here, $Q_{\text{CPU},p,B}^{\text{agg}}$ is not the actual elapsed time of a 200-batch CPU block measured with a separate outer timer; it is a derived sensitivity metric computed by summing the existing per-batch timings. Accordingly, this analysis does not reconstruct identical CPU and CPU–GPU measurement boundaries; it asks how much the primary comparison changes when only the CPU block-aggregation rule is changed.

4.3 Correctness Validation

Performance measurement for an execution path begins only after that path passes its predefined correctness validation. We distinguish development-stage validation of GPU-assisted grinding from formal validation bound directly to the final performance-measurement paths; Table 5 summarizes the scope and role of each validation layer. The candidate bitmap itself is checked separately through a development microcost validation.

Table 5. Correctness-validation layers and coverage

Validation layer	Coverage	Checks	Role
Development-stage grinding-semantics validation	10 signatures for each of six parameter sets, 60 total; 20 signatures for FAEST-128s and FAEST-192s actually use GPU-assisted grinding	Selected counter, minimum accepting counter, final challenge, BAVC-opening result, complete signature bytes, current implementation verification, and faest-ref verification	Check whether GPU preliminary filtering plus CPU ordered acceptance preserves the grinding semantics of the reference path
Final M0–M3 formal validation	4 measurement methods $\times$ 6 parameter sets = 24 configurations; $100 + 1000 = 1100$ input rows per configuration	Complete signature bytes, grinding counter, current implementation verification, and faest-ref verification	End-to-end correctness of the actual latency and diagnostic paths bound to the final source and binaries
Final throughput formal validation	6 parameter sets $\times$ 3 batch sizes = 18 configurations; one B -request batch per configuration	Per-request signature bytes and counter, input–output mapping, current implementation verification, and faest-ref verification	Check that overlapping multiple requests does not mix results and returns the correct signature for each request

In the development microcost validation, for each of the three candidate chunk sizes tested for FAEST-128s and FAEST-192s, the GPU candidate bitmap is compared directly on a per-chunk basis with a reference bitmap independently computed by the 4-way CPU hash path. No bitmap mismatch is observed across the 102 chunk executions recorded from 60 development-smoke rows or the 17,656 chunk executions recorded from 12,000 microcost rows. These chunk counts are sums of recorded executions rather than counts of unique counter ranges, and this evidence applies to the development microcost implementation rather than replacing the formal validation bound to the final integrated implementation.

The formal-validation coverage for M0–M3 and throughput is given in Table 5; any mismatch would prevent performance measurement from starting for the corresponding execution path.

For M0–M3 validation, the complete signature byte string and grinding counter generated from the same key, message, and signing randomness ρ are compared with predetermined expected results. Each signature is checked by the current implementation’s verification procedure and by the separate **faest-ref** reference implementation, thereby checking agreement of the deterministically generated result in addition to successful return.

Development-stage semantic validation of GPU-assisted challenge grinding uses 10 signatures for each of the six parameter sets, for 60 signatures in total; among these, the 20 FAEST-128s and FAEST-192s signatures actually execute the GPU-assisted grinding path. For each case, we compare the selected counter, minimum accepting counter, final challenge, BAVC-opening result, and complete signature bytes with the CPU reference path; all compared values match, and both the current implementation and **faest-ref** verification succeed. This result provides development-stage evidence that GPU candidate filtering plus final CPU acceptance in increasing order preserves c^* from Section 2.1 for the tested executions.

Formal validation before final performance measurement does not replace development-stage semantic validation; rather, it confirms the end-to-end correctness of the actual measurement paths bound to the final source and binaries. Thus, development-stage validation supports the internal grinding semantics, while formal validation supports correctness of the signatures generated by the final measurement paths.

The multi-sign pipeline is validated separately for the 18 configurations formed by six parameter sets and $B \in \{2, 4, 8\}$. For every request, we check the signature byte string and grinding counter, the input–output mapping, and successful verification by both the current implementation and **faest-ref**, confirming that overlapping requests does not mix their results and that each request returns the correct signature.

All 24 M0–M3 configurations and 18 throughput configurations bound to the final performance evaluation, 42 configurations in total, pass both the current implementation and **faest-ref** verification, with no configuration rejected by correctness validation. Performance results are therefore reported only for paths that pass the predefined correctness conditions, and inputs and signatures used for correctness validation are not treated as performance measurements.

These checks establish implementation-level agreement for the evaluated input corpus and execution configurations; they are not exhaustive verification over all possible inputs and counters and do not replace the security proof of FAEST v3. They also do not establish constant-time behavior or side-channel security for the complete implementation; our correctness claim is limited to agreement with the reference path over the evaluated scope and successful verification by both the current implementation and a separate reference implementation.

4.4 Diagnostic Validation and Measurement Integrity

M2 and M3-B2 observe the same logical `fc_run` boundary defined in Section 4.2, respectively in a separate diagnostic execution and within one complete signing call. Although both target the same function boundary on the production path, they run in different execution contexts and are therefore not treated as repeated measurements of the same invocation or as interchangeable values.

Let $C_{M2,p}$ and $C_{M3-B2,p}$ denote the representative M2 and M3-B2 values, respectively. We define their signed relative difference as

$$D_p = \frac{C_{M3-B2,p} - C_{M2,p}}{C_{M2,p}} \times 100. \quad (15)$$

Table 6 reports the observed relative differences across the six parameter sets.

Table 6. Comparison of representative M2 and M3-B2 measurements

Parameter	M2 (ns)	M3-B2 (ns)	D_p (%)
FAEST-128s	433,613.5	434,723.5	0.256
FAEST-128f	117,241.5	118,185.5	0.805
FAEST-192s	1,086,065.5	1,077,903.5	-0.752
FAEST-192f	235,219.0	240,235.5	2.133
FAEST-256s	2,126,049.0	2,122,086.0	-0.186
FAEST-256f	359,116.0	359,725.5	0.170

The absolute relative differences range from approximately 0.17–2.13%, and their signs vary by parameter set. Because no equivalence threshold was defined in advance, we use these values only as diagnostic observations of the same production boundary under different execution contexts and do not claim statistical equivalence or interchangeability of the two measurements.

Across the three diagnostic sessions and six parameter sets, M3 records B1, B2, B3, the sample identifier, and the key identifier together within the same signing call for all 18,000 measured rows. There are no missing B1/B2/B3 values, duplicate sample identifiers, or rows combining values from different calls, and each M3-B2 measurement invokes both the `fc_run` wrapper and the actual function exactly once. No invalid M3-B1 CUDA event or row with B2 longer than B3 is observed, confirming that B1, B2, and B3 were recorded from the same signing call rather than combined post hoc from independent executions.

Measurement eligibility is determined by conditions defined before execution rather than by the magnitude of individual measurements. If GPU activity from another process is detected before a GPU measurement, the block is not started; if such activity is detected during a session, the entire session is marked ineligible rather than removing individual measurements. Long execution times, high-percentile latency, temperature or clock changes, and ordinary system load are not by themselves classified as contamination. Only documented external

contamination permits at most one replacement of the entire session; otherwise, automatic reruns, selective repetition after inspecting results, post-hoc outlier removal, and arbitrary addition of missing rows are disallowed. No session in the final formal evaluation was marked ineligible or replaced because of external GPU activity.

Each primary-latency and diagnostic block included in the results must contain exactly 1000 valid measurement rows, and each throughput block must contain all 200 measured batches. Every M3 row must contain valid B1, B2, and B3 values with B2 not exceeding B3, and rows or batches are not selectively excluded based on the observed results.

In the final evaluation, primary latency and diagnostic measurements each consist of 36 blocks and 36,000 measured rows, while throughput consists of 72 blocks and 14,400 batch-measurement rows containing 67,200 individual signature executions in total. Thus, eligibility checks cover 144 blocks and 86,400 measured rows in total; the 14,400 throughput values are batch-level measurement rows rather than individual signatures. All 144 expected blocks are present and pass the eligibility checks, with no missing block or duplicate eligibility identifier.

Variation across repeated measurements is described by the relative range, defined as the difference between the maximum and minimum representative values under the same condition divided by the final representative value. The maximum relative ranges are 1.791% and 1.785% for M0 and M1; 1.152%, 0.013%, 0.532%, and 0.402% for M2, M3-B1, M3-B2, and M3-B3; and 2.541% and 3.379% for the CPU and CPU-GPU hybrid throughput paths, respectively. Because no pass threshold was defined in advance, these values are descriptive statistics of between-run variation and do not establish statistical independence or passage of a predefined stability criterion.

5 Performance Results

5.1 Single-Sign Latency

Table 7 reports the representative single-sign latencies of M0 and M1 and the comparison ratio R_p for the six parameter sets. Representative values are aggregated as defined in Section 4.2; because R_p aggregates the within-session ratios of the M0 and M1 medians, it is not generally equal to $L_{0,p}/L_{1,p}$.

We define the relative change in representative M1 latency as

$$\Delta_p = 100 \frac{L_{1,p} - L_{0,p}}{L_{0,p}}. \quad (16)$$

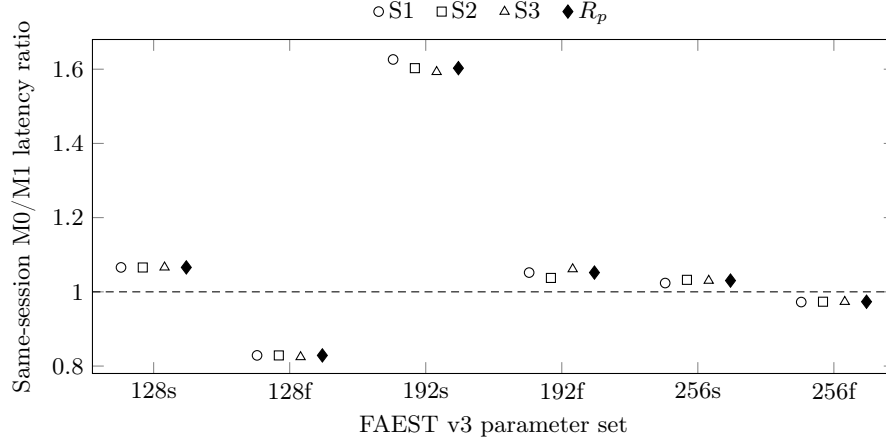
A value $\Delta_p < 0$ indicates the direction in which M1 has lower representative latency, whereas $\Delta_p > 0$ indicates the direction in which M0 is lower. The value Δ_p is derived from the representative latencies $L_{0,p}$ and $L_{1,p}$ and is not a percentage transformation of the same-session ratio aggregate R_p .

Figure 4 shows the same-session M0/M1 median ratio for S1, S2, and S3 for each parameter set together with their median R_p . Each point is computed from

Table 7. Single-sign latency for the six FAEST v3 parameter sets

Parameter	M0 (μs)	M1 (μs)	R_p (M0/M1)	Δ_p (%)
FAEST-128s	2,321.2365	2,177.3485	1.065855	-6.20
FAEST-128f	373.0515	451.4145	0.828695	+21.01
FAEST-192s	10,338.2665	6,457.7385	1.602799	-37.54
FAEST-192f	1,722.0860	1,634.4865	1.051903	-5.09
FAEST-256s	14,975.9770	14,539.2840	1.030325	-2.92
FAEST-256f	2,227.2410	2,288.5090	0.973353	+2.75

the medians of 1000 measurements within a session rather than from ratios of raw measurement rows, and the horizontal dashed line marks ratio 1.

**Fig. 4.** Session-wise M0/M1 latency ratios and representative comparison ratio

Among the six parameter sets, M1 has lower representative latency than M0 for FAEST-128s, FAEST-192s, FAEST-192f, and FAEST-256s. Conversely, M1 has higher representative latency for FAEST-128f and FAEST-256f. The direction of the single-sign result is therefore not consistent across the six parameter sets.

The largest relative difference in favor of M1 is observed for FAEST-192s. Its representative latency decreases by 37.54%, from 10,338.2665 μs for M0 to 6,457.7385 μs for M1, and the representative same-session ratio is $R_{192s} = 1.602799$.

Results in the opposite direction are observed for FAEST-128f and FAEST-256f; of these, FAEST-128f has the larger difference, with M1 representative latency 21.01% higher than M0 and $R_{128f} = 0.828695$.

Even at the same nominal security level, the s and f parameter sets do not always exhibit the same direction, and no monotonic trend is observed in the

relative M0/M1 difference as the security level increases across the six measured parameter sets.

Because M1 in Table 7 measures the complete CPU–GPU hybrid signing call under the parameter-specific execution policy of Section 3.1, we do not interpret the M0–M1 difference as the independent effect of challenge grinding alone.

5.2 Multi-Sign Throughput

Table 8 reports the representative throughput and ratio $G_{p,B}$ of the CPU baseline and CPU–GPU hybrid paths for the six parameter sets and $B \in \{2, 4, 8\}$. Throughput is the number of independent FAEST signatures completed per second, and aggregation and the definition of $G_{p,B}$ follow Section 4.2. Throughput values in the table are rounded to three decimal places, while $G_{p,B}$ is computed from the unrounded representative values.

Table 8. Representative throughput for multiple independent signatures

Parameter	B	CPU (sig/s)	CPU–GPU hybrid (sig/s)	$G_{p,B}$ (hybrid/CPU)
FAEST-128s	2	756.329	772.732	1.021687
	4	1,353.563	1,255.565	0.927600
	8	1,656.031	1,372.235	0.828629
FAEST-128f	2	5,006.734	4,906.406	0.979962
	4	9,566.703	8,350.906	0.872914
	8	10,570.535	10,040.107	0.949820
FAEST-192s	2	174.226	194.274	1.115069
	4	308.381	308.618	1.000770
	8	483.848	454.106	0.938531
FAEST-192f	2	1,103.148	1,290.553	1.169882
	4	2,130.596	2,350.576	1.103248
	8	2,918.416	2,976.630	1.019947
FAEST-256s	2	148.559	164.171	1.105087
	4	294.067	287.252	0.976825
	8	402.164	342.233	0.850980
FAEST-256f	2	872.303	946.645	1.085225
	4	1,716.147	1,735.995	1.011565
	8	2,324.525	2,196.268	0.944825

Figure 5 compares representative throughput ratios $G_{p,B}$ for $B = 2, 4, 8$ within each parameter set. The three batch sizes are distinct discrete measurement conditions, and the horizontal dashed line marks ratio 1.

Under the predefined primary aggregation rule, the CPU–GPU hybrid path has higher representative throughput in 9 of the 18 parameter–batch-size combinations, while the CPU baseline is higher in the remaining 9. FAEST-128f

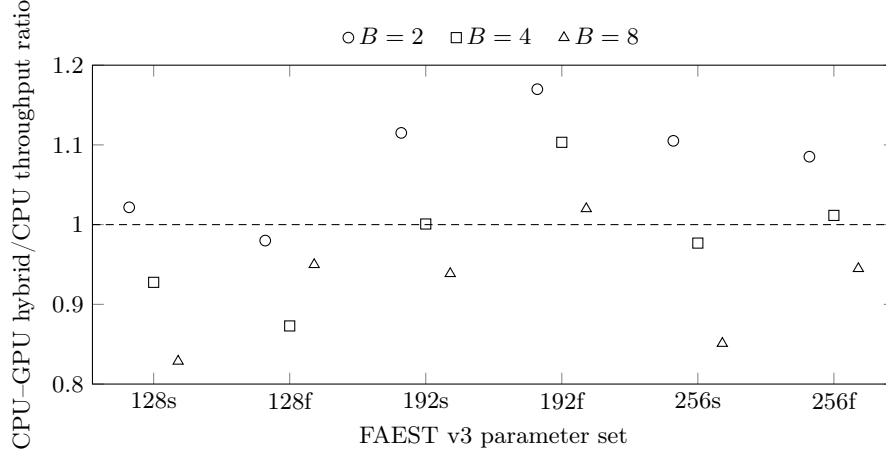


Fig. 5. CPU–GPU hybrid/CPU throughput ratios by batch size

favors the CPU baseline for all three measured batch sizes, FAEST-192f favors the CPU–GPU hybrid path for all three, and the relative direction changes with B for the remaining four parameter sets.

In the post-hoc auxiliary aggregation, 17 of the 18 conditions retain the same comparison direction as the primary aggregation. The exception is FAEST-128f at $B = 2$, where $G_{128f,2} = 0.979962$ changes to $G_{128f,2}^{\text{agg}} = 1.016331$, reversing the comparison direction. The largest relative change across all conditions is approximately 5.98%, observed for FAEST-128s at $B = 4$; these auxiliary results do not replace the representative throughputs or ratios of the primary protocol.

The maximum hybrid/CPU ratio is $G_{192f,2} = 1.169882$ for FAEST-192f at $B = 2$, where the representative throughput of the CPU–GPU hybrid path is approximately 16.99% higher. The minimum is $G_{128s,8} = 0.828629$ for FAEST-128s at $B = 8$, where the representative throughput of the CPU–GPU hybrid path is approximately 17.14% lower.

For all six evaluated parameter sets, $B = 2$ gives the largest $G_{p,B}$ among the three measured batch sizes, but the ratio does not change monotonically with increasing B for every parameter set. Thus, over the measured range $B \in \{2, 4, 8\}$, we do not observe a consistent monotonic relationship between increasing batch size and the relative advantage of the CPU–GPU hybrid path.

For FAEST-192s at $B = 4$, $G_{192s,4} = 1.000770$ is numerically the closest to 1 among the 18 conditions; because no equivalence criterion was defined in advance, we interpret it only as a value close to 1 rather than as evidence of performance equivalence.

6 Discussion and Limitations

Taken together, Sections 5.1 and 5.2 show that the relative effect of CPU–GPU hybrid execution varies with both the FAEST v3 parameter set and the execution condition. For single signatures, M1 has lower representative latency for four of the six parameter sets; under the predefined primary aggregation rule for throughput, the CPU–GPU hybrid path has higher representative throughput in 9 of the 18 conditions. Accordingly, no consistent CPU–GPU performance advantage common to the entire evaluated scope is observed.

Results can also differ between single-sign and multi-sign processing for the same parameter set. FAEST-192s has $R_{192s} = 1.602799$ in the single-sign evaluation, whereas its throughput ratios are $G_{192s,B} = 1.115069, 1.000770$, and 0.938531 for $B = 2, 4, 8$, respectively. Thus, the relative difference observed for a single signature cannot be generalized as having the same direction and magnitude when multiple requests are processed together.

In the throughput evaluation, $B = 2$ produces a larger $G_{p,B}$ than $B = 4$ and $B = 8$ for all six measured parameter sets, but the change in the ratio as B increases is not monotonic for every parameter set. Moreover, the post-hoc auxiliary analysis that changes the CPU block-aggregation rule reverses the comparison direction in one of the 18 conditions. We therefore do not infer a general monotonic relationship between batch size and the relative advantage of CPU–GPU hybrid execution over the measured $B \in \{2, 4, 8\}$ range, nor do we generalize the 9-to-9 split under the primary aggregation as a property independent of the aggregation method.

The CPU baseline in the throughput comparison processes the same B independent requests as the CPU–GPU hybrid path. Thus, $G_{p,B}$ compares representative throughput of the two execution paths at the same request count rather than measuring the absolute scalability of the GPU pipeline itself. We do not infer the relative relationship between the two paths for larger values of B that were not measured.

The single-sign and throughput evaluations differ not only in workload but also in the actual execution path. At $B = 1$, FAEST-128s and FAEST-192s use GPU-assisted challenge grinding while the other four parameter sets retain 4-way CPU grinding; at $B > 1$, all six parameter sets retain 4-way CPU grinding and the GPU instead executes the batched work. Accordingly, the single-sign ratio R_p and throughput ratio $G_{p,B}$ are not repeated measurements of the same optimization under different loads and cannot be combined into a single common performance metric.

These results indicate that GPU use for FAEST v3 signing should be evaluated separately for the target parameter set and expected request pattern rather than treated as one global decision across the entire evaluated space. However, we do not propose or evaluate a general runtime dispatch policy that automatically selects between CPU and GPU paths based on these measurements.

The first limitation is that the evaluation environment is restricted to a single hardware configuration. All formal performance results were measured on the same system containing an Intel Core i7-13700KF and an NVIDIA GeForce RTX

4080, so the same relative results cannot be generalized to other CPU/GPU combinations, host-GPU interconnects, or software environments. In addition, the CPU worker threads in the throughput evaluation are not pinned to specific logical CPUs or P/E-cores, so the effect of worker-placement policy on the throughput comparison is not isolated.

The second limitation is that throughput is evaluated only for $B \in \{2, 4, 8\}$. These results show relative throughput only at three measured points and do not determine performance for $B > 8$ or whether the relative direction of the two paths changes again. The measurements therefore do not establish the batch size that maximizes throughput or a general crossover point between the two execution paths.

The third limitation concerns implementation scope. We evaluate only the signing path of the six non-EM FAEST v3 parameter sets, excluding GPU acceleration of verification, multi-GPU execution, and multi-stream designs that execute multiple GPU batches concurrently on one GPU. Because $B > 1$ retains 4-way CPU grinding, the results also do not cover a multi-sign design that combines GPU-assisted challenge grinding with the GPU pipeline.

The fourth limitation is that we do not isolate the causes of the observed performance differences at the level of individual components. The formal performance evaluation compares the complete CPU-GPU hybrid path actually used with the CPU baseline and does not perform formal ablations that individually remove pinned memory, slot alternation, a specific GPU kernel, or challenge grinding. We therefore do not attribute the observed end-to-end performance difference to the independent causal effect of any single component. M2 and M3 in Section 4.4 are likewise diagnostics for confirming the intended measurement boundaries rather than measurements for decomposing performance causes. Development-stage microbenchmarks and profiling informed implementation decisions but do not replace the formal performance results in Sections 5.1 and 5.2.

The fifth limitation is the scope of the evaluation metrics. We evaluate single-sign latency and throughput for multiple independent signatures but do not measure energy consumption, power efficiency, or cost efficiency. Consequently, the energy efficiency or economic efficiency of GPU use cannot be judged from these results.

Finally, correctness validation is intended to establish implementation-level agreement for the evaluated source, binaries, input corpus, and execution configurations. The validation results in Section 4.3 support that the CPU and CPU-GPU paths produce valid FAEST signatures within the evaluated scope, but they do not establish formal correctness over all possible inputs or constant-time behavior of the complete implementation. We therefore do not extend these results into a claim of side-channel security or implementation security for an entire production environment.

7 Conclusion

We implemented two CPU–GPU hybrid execution paths for FAEST v3 signing and compared their performance with an optimized CPU baseline on the same system. For single signatures, the GPU filters preliminary challenge-grinding candidates in parallel while the CPU performs final acceptance checks in increasing counter order, preserving the minimum-accepting-counter semantics. For multiple independent signatures, two signing-state slots and one shared GPU execution resource overlap the GPU batch of one slot with CPU work for the other. Neither execution path changes the FAEST signature format or verification procedure.

For single signatures, the CPU–GPU hybrid path M1 has lower representative latency for four of the six non-EM parameter sets, whereas the CPU baseline M0 is lower for the other two. The largest representative comparison ratio is $R_{192s} = 1.602799$ for FAEST-192s, where representative M1 latency is 37.54% lower than M0. Thus, no single-sign performance improvement common to all evaluated parameter sets is observed.

For throughput of multiple independent signatures, we compare 18 conditions formed by $B \in \{2, 4, 8\}$ and the six parameter sets; under the predefined primary aggregation rule, the CPU–GPU hybrid path has higher representative throughput in 9 conditions and the CPU baseline in the remaining 9. For some parameter sets, the relative direction between the two execution paths also changes with B . Accordingly, the throughput results likewise do not show one common direction across all evaluated conditions.

Because single-sign and multi-sign processing use different execution paths and measurement boundaries, we do not combine their results into one common speedup. The formal evaluation measures each complete CPU–GPU hybrid path, so the observed differences are not attributed to the independent causal effect of any one component. Development-stage validation of GPU-assisted challenge grinding confirms agreement for the minimum accepting counter, final challenge, BAVC-opening result, and complete signature, and all 42 execution configurations bound to the final performance evaluation pass both the current implementation’s verification procedure and separate **faest-ref** verification.

The evaluation is limited to one CPU–GPU system, the six non-EM parameter sets, and a limited set of throughput conditions; other hardware configurations, larger batch sizes, GPU-assisted verification, and concurrent execution of multiple GPU batches are not evaluated. Within this scope, the results show that both the direction and magnitude of the relative performance effect of GPU assistance for FAEST v3 signing can vary with the parameter set and execution condition.

References

1. Baum, C., Beullens, W., Braun, L., Delpech de Saint Guilhem, C., Kloof, M., Majenz, C., Mukherjee, S., Orsini, E., Ramacher, S., Rechberger, C., Roy, L., Scholl, P.: Shorter, tighter, FAESTer: Optimizations and improved (QROM) analysis for VOLE-in-the-Head signatures. In: Advances in Cryptology – CRYPTO 2025.

- Lecture Notes in Computer Science, vol. 16005, pp. 124–156. Springer (2025). https://doi.org/10.1007/978-3-032-01887-8_5
2. Baum, C., Beullens, W., Braun, L., Delpech de Saint Guilhem, C., Klooß, M., Majenz, C., Mukherjee, S., Orsini, E., Ramacher, S., Rechberger, C., Roy, L., Scholl, P.: FAEST v3: Algorithm specifications. Version 3.0 (2026), <https://faest.info/faest-spec-v3.0.pdf>, September 1, 2026
 3. Baum, C., Beullens, W., Mukherjee, S., Orsini, E., Ramacher, S., Rechberger, C., Roy, L., Scholl, P.: One tree to rule them all: Optimizing GGM trees and OWFs for post-quantum signatures. In: Advances in Cryptology – ASIACRYPT 2024. Lecture Notes in Computer Science, vol. 15484, pp. 463–493. Springer (2024). https://doi.org/10.1007/978-981-96-0875-1_15
 4. Baum, C., Braun, L., Delpech de Saint Guilhem, C., Klooß, M., Orsini, E., Roy, L., Scholl, P.: Publicly verifiable zero-knowledge and post-quantum signatures from VOLE-in-the-Head. In: Advances in Cryptology – CRYPTO 2023. Lecture Notes in Computer Science, vol. 14085, pp. 581–615. Springer (2023). https://doi.org/10.1007/978-3-031-38554-4_19
 5. FAEST software. <https://faest.info/software.html>, accessed September 27, 2026
 6. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) Advances in Cryptology – CRYPTO ’86. Lecture Notes in Computer Science, vol. 263, pp. 186–194. Springer (1987). https://doi.org/10.1007/3-540-47721-7_12
 7. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. *Journal of the ACM* **33**(4), 792–807 (1986). <https://doi.org/10.1145/6490.6503>
 8. National Institute of Standards and Technology: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. Federal Information Processing Standards Publication 202, National Institute of Standards and Technology (2015). <https://doi.org/10.6028/NIST.FIPS.202>
 9. National Institute of Standards and Technology: Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, National Institute of Standards and Technology (2023). <https://doi.org/10.6028/NIST.FIPS.197-upd1>, updated May 9, 2023
 10. Ning, Y., Dong, J., Lin, J., Zheng, F., Fu, Y., Xiao, F.: GRASP: Accelerating hash-based PQC performance on GPU parallel architecture. *IEEE Transactions on Computers* **75**(4), 1579–1592 (2026). <https://doi.org/10.1109/TC.2026.3654650>
 11. NVIDIA Corporation: CUDA C++ Programming Guide. NVIDIA Corporation (2023), <https://docs.nvidia.com/cuda/archive/12.0.1/cuda-c-programming-guide/>, CUDA Toolkit 12.0.1, archived documentation, accessed September 27, 2026
 12. Shen, S., Yang, H., Dai, W., Zhang, H., Liu, Z., Zhao, Y.: High-throughput GPU implementation of Dilithium post-quantum digital signature. *IEEE Transactions on Parallel and Distributed Systems* **35**(11), 1964–1976 (2024). <https://doi.org/10.1109/TPDS.2024.3453289>
 13. Yang, K., Sarkar, P., Weng, C., Wang, X.: QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 2986–3001. ACM (2021). <https://doi.org/10.1145/3460120.3484556>