Heterogeneous CPU–NPU Implementation of the AIMer Post-Quantum Signature for Low Latency on a Mobile SoC

Myoungsu Shin[1], Jihoon Jang[2], Hanbeom Shin[2], Insung Kim[2], Dongjae Lee[3] and Donggeun Kwon[4] *

[1] Independent Researcher, Seoul, Republic of Korea houma757@gmail.com
[2] Korea University, Seoul, South Korea {jhw1031506,newonetiger,cmcom35}@korea.ac.kr
[3] Kangwon National University, Chuncheon, Republic of Korea, dongjae.lee@kangwon.ac.kr
[4] Kunsan National University, Gunsan, South Korea, dgkwon@kunsan.ac.kr

Abstract. Signatures in the MPC-in-the-head paradigm rest only on a symmetric primitive, but their signing is slow, because it computes hash functions and a linear map for every simulated party of every repetition. We show that the Hexagon NPU of Snapdragon SoCs shortens one complete signing operation of AIMer, the MPC-in-the-head signature that the Korean post-quantum cryptography competition selected. Prior work speeds up such signatures on one CPU core, accelerates post-quantum arithmetic on GPUs for batches of operations, and reports only single post-quantum functions for mobile NPUs. Our implementation divides each signing operation along its dependency graph. The NPU computes the hashing and the per-party linear map of the last repetitions, which need no secret key, and the CPU keeps the secret key and the two hashes that absorb the repetitions in index order. A fast AArch64 implementation serves both as the CPU part and as the baseline, and it signs 4.5 to 6.1 times faster than the reference implementation on one core. On a Snapdragon 8 Gen 3 tablet, one core with the NPU signs 1.30 to 1.42 times faster than two cores and 1.74 to 1.97 times faster than one core for the three short-signature parameter sets. The same configuration uses 13 to 27 % less energy per signing operation than two cores. On Snapdragon 8 Elite and 8 Gen 2 phones, one core with the NPU signs 1.28 to 1.35 and 1.74 to 1.82 times faster than two cores.

Keywords: MPC-in-the-head signatures · heterogeneous CPU–NPU computing · AIMer · Hexagon HVX

1 Introduction

AIMer [CHK+26] is one of the two signatures that the Korean Post-Quantum Cryptography Competition (KpqC) selected [Kpq26]. It follows the paradigm of multi-party computation (MPC) in the head [IKOS07, CDG+17]. The signer simulates an MPC of the one-way function AIM3 on the secret key, and the signature opens the views of all parties but one. The security of such signatures rests only on a symmetric primitive, but the price is computation. Every party of every repetition needs its own SHAKE hashes and its own evaluation of a binary linear map for each party, and a signing operation of the short-signature parameter sets simulates 256 parties in 17 to 33 repetitions.

Post-quantum signatures are also moving onto phones, and Android 17 adds native support for ML-DSA to the Android Keystore [LE26]. Post-quantum signatures slow the pro-

*Corresponding author.

tocols that carry them through their size and computation [SKD20], and NIST judges the MPC-in-the-head candidates PERK and SDitH below ML-DSA in performance [A+24].

Implementations of post-quantum schemes usually reduce such costs on one CPU core with data-parallel code, for lattice schemes [KMRV18, BHK+22] as well as for MPC-in-the-head signatures. Accelerator work moves polynomial arithmetic to the Tensor Cores of GPUs, which gain throughput when one batch holds many independent operations, and to TPUs. The related work at the end of this section compares these implementations with ours.

Mobile SoCs also integrate a neural processing unit (NPU) next to their CPU cores. For smartphones with a built-in NPU or neural engine that runs small language models, Gartner forecasts end-user sales of 487 million units in 2026, up from an estimated 369 million in 2025 [Gar25]. In contrast to GPUs, the mobile NPU has received little attention in cryptography. For the Hexagon NPU of Qualcomm SoCs, prior post-quantum work reports only single functions on a simulator [Cau25] and batched HQC decoding [CKM+26]. To our knowledge, no prior work uses a mobile NPU to shorten one complete signing operation.

Shortening one operation is harder than raising throughput. Each CPU–NPU message has a fixed cost that no other operation shares, and parts of a signing operation depend on the secret key or on hashes that absorb their inputs in order. An NPU therefore helps only if it computes a large part of the remaining work while the CPU computes the rest.

In this work, we split one AIMer signing operation between the CPU cores and the Hexagon NPU of Qualcomm Snapdragon SoCs. We first build a fast implementation for the CPU, because a slow CPU baseline overstates the gain of any accelerator, and because this code also computes the part of the operation that stays on the CPU, which we call the *host*. We then move the work that needs no secret key to the NPU. Our contributions are as follows.

- **AArch64 implementation** (Section 3). The host code applies three techniques, one for each large cost of a repetition. The grouped MPC of Kwon et al. [KLL+25], ported from AIM2 to AIM3, handles the field arithmetic, a bit-sliced subset-XOR multiplication handles the linear map of all 256 parties, and the two-way NEON Keccak of mlkem-native [The26, BK22] handles the SHAKE hashes. The host code signs 4.5 to 6.1 times faster than the reference implementation [Sam26] (Ref).
- **NPU kernels** (Section 4). The NPU computes the SHAKE hashes and the linear map, which do not read the secret key. The Hexagon Vector eXtensions (HVX) of the compute DSP (cDSP) have 1024-bit vectors, and our 16-way Keccak keeps all 16 states in the 32 vector registers, one lane per vector. Four HVX units then compute the SHAKE hashes of one repetition 1.77 to 1.94 times faster than one Cortex-X4 core, in a message of five repetitions. One vector also holds a bit plane of four repetitions for the linear map. Because a shared buffer keeps each further message at 8 to 22 µs, the host can split the NPU work into several messages and use each result while the NPU computes the next.
- **Work partitioning** (Section 5). From the dependency graph of a signing operation, we derive which work the NPU can compute without the secret key. We give the NPU the SHAKE hashes and the linear map of the last repetitions, and the CPU keeps the secret key, the last party, and the two hashes that absorb their inputs in a fixed order. A search on each device selects how many repetitions move, and the same partition also accelerates verification.

With the NPU, one CPU core signs faster than two CPU cores on all three SoCs that we evaluate (Section 6). The evaluation covers signing and verification on one tablet and two phones, with inputs that no search uses, and its strict comparison counts the NPU like one more CPU core. On the Snapdragon 8 Gen 3 (SM8650) tablet, one core with the NPU signs 1.30 to 1.42 times faster than two cores and 1.74 to 1.97 times faster than

one core (Table 8). It also verifies 1.20 to 1.33 times faster than two cores, and on the battery of the tablet it uses 13 to 27 % less energy per signing operation than two cores (Table 10). On a Snapdragon 8 Elite (SM8750) and a Snapdragon 8 Gen 2 (SM8550) phone, the speedup of signing over two cores is 1.28 to 1.35 and 1.74 to 1.82.

Related work. Implementations of MPC-in-the-head signatures usually target one processor. The Picnic implementation of Kales and Zaverucha computes the commitments of the parties four at a time with an AVX2 Keccak [KZ20], PERK and MQOM are optimized for embedded processors [BBB+24, BF26], and SDitH in Hardware maps an MPC-in-the-head signature to FPGA logic [DHSY24]. MQOM also bit-slices the parallel repetitions of its proof [BF26], and our NPU kernel applies this principle to four repetitions in one HVX vector. For AIMer with AIM2, Kwon et al. group the per-party field operations on AArch64 and AVX2 [KLL+25], and Lee et al. vectorize them on AVX-512 [LES26]. For AIM3, Lee et al. do so on AVX2, AVX-512, and the Cortex-M55 [LEK+26]. For AIM-I, the 128-bit parameter set of the original AIM, Lee et al. combine Mersenne exponentiations and precompute the matrices of one fixed initialization vector [LJK+23]. Our host code precomputes these matrices per key and bit-slices the linear map over all 256 parties, which the MPC implementations above compute for each party.

Accelerators for post-quantum cryptography mostly speed up one kernel or a batch of operations. ConvKyber and TensorCrypto map the polynomial arithmetic of lattice schemes to NVIDIA Tensor Cores [ZZF+24, LSZH22], and Gazzoni Filho et al. use the matrix unit of Apple SoCs for NTRU [GBL24]. Karanjai et al. use TPUs for the polynomial multiplication of homomorphic encryption and zero-knowledge proofs [KSX+23]. On the Hexagon DSP, Avanzi and Brumley accelerate the Long-Term Evolution (LTE) algorithms 128-EEA3 and 128-EIA3 with delayed modular reduction and, for 128-EIA3, with carryless multiplication [AB15]. Caullery measures Keccak, the NTT of Kyber, and the sparse multiplication of HQC-128 with HVX on a cycle-accurate Hexagon simulator [Cau25], and Chau et al. batch many HQC decodes behind each FastRPC call [CKM+26]. In contrast to these works, we accelerate one complete signing operation, and the cost of every host–NPU message therefore stays inside the measured latency.

Closest to our goal, two works divide one signing operation between processors. Banegas and Caullery divide a SPHINCS+ signing operation over the application cores of a Snapdragon 865 and their NEON units [BC23]. Kim et al. divide a FAEST v3 signing operation between a desktop CPU and GPU, and the divided operation is faster for four of the six parameter sets without the Even–Mansour construction [KEL+26]. We instead divide the operation along its dependency graph between the CPU and a mobile NPU. Other CPU–GPU systems target the throughput or energy efficiency of independent requests or bulk data [JHH+11, LSKY19, FLYL20, WDX+25].

Section 2 describes AIMer, the dependencies of one signing operation and of one repetition, and the platform. Section 3 and Section 4 describe the host and the NPU sides, and Section 5 derives the partition of the operation between them. Section 6 evaluates signing and verification, and Section 7 concludes.

Code availability. The source code and the build scripts are available at `https://anonymous.4open.science/r/AIMer-CPU_NPU_hetero-Qualcomm-anonymous` under an open license. The implementation builds on the AIMer reference implementation at commit `734cc76` [Sam26] and on the Hexagon SDK 6.5.0.0 [Qua25].

2 Preliminaries

This section describes AIMer and AIM3, the five phases of one signing operation, which the rest of the paper divides between the CPU and the NPU, and the target platform.

Table 1: AIMer parameter sets [CHK+26].

					Size (bytes)		
Parameter set	n	L	N	T	$\|pk\|$	$\|sk\|$	$\|sig\|$
128f	128	2	16	33	32	48	6 944
192f	192	2	16	49	48	72	15 408
256f	256	3	16	65	64	96	31 360
128s	128	2	256	17	32	48	4 704
192s	192	2	256	25	48	72	10 320
256s	256	3	256	33	64	96	20 224

2.1 MPC-in-the-Head and AIMer

In the MPC-in-the-head paradigm, a prover simulates a multi-party computation among N parties that checks a relation on a secret-shared witness, commits to the views of all parties, and opens the views of the parties that the verifier selects [IKOS07]. In KKW and in AIMer, the relation is the evaluation of a one-way function on the secret key, and the verifier selects all parties but one [KKW18, CHK+26]. The opened views convince the verifier that the computation is correct, and the missing view keeps the witness secret. Picnic showed that this paradigm gives practical post-quantum signatures [CDG+17], KKW reduced the proof size with a preprocessing phase and more parties [KKW18], and the BN++ proof system [KZ22] refines the protocol of Baum and Nof for arithmetic circuits over large fields [BN20]. The one-way functions AIM and AIM2 of AIMer use few multiplications over a large binary field [KHS+23, KHSL23], and after the cryptanalysis of AIM2 [BFU26], version 3.0 replaces AIM2 with AIM3 [CHK+26].

AIMer applies the Fiat–Shamir transform to a customized BN++ proof of knowledge of a preimage of AIM3, and this paper implements version 3.0 of its specification [CHK+26]. Key generation samples a field element pt and a public initialization vector iv, rejects an instance that has a zero S-box input, and publishes (iv, ct) with $ct = \mathrm{AIM3}(pt, iv)$. The preimage pt is the signing witness, and during signing $N = 256$ simulated parties hold additive shares of pt over $\mathbb{F}_{2^n}$. Table 1 lists the six parameter sets. This paper evaluates the three short-signature sets (s), which use $N = 256$ parties and $T = 17$, 25, or 33 repetitions, and the hash is SHAKE128 at the 128-bit level and SHAKE256 at the 192-bit and 256-bit levels. The fast sets (f) are outside the scope, because they give the NPU little work for the fixed cost of each call to the NPU: 128f signing holds 1 023 hashes against 8 687 for 128s (Subsection 4.1 and Subsection 4.2).

2.2 The AIM3 One-Way Function

AIM3 [CHK+26] maps pt to ct with L input S-boxes, one output S-box, and two affine layers. For the number L of input S-boxes, the public vector iv expands through an extendable-output function (XOF) into $2L$ invertible binary matrices $A_0, \ldots, A_{2L-1}$ and offsets $b_0, \ldots, b_L$. For $0 \le \ell < L$, AIM3 computes

$$x_\ell = A_\ell\, pt + b_\ell, \qquad y_\ell = S_{e_\ell}(x_\ell),$$

and it combines the branches with one output S-box:

$$x_L = \sum_{\ell=0}^{L-1} A_{L+\ell}\, y_\ell + b_L, \qquad ct = pt + S_{e_L}(x_L).$$

All additions are XORs in the polynomial basis. The enhanced Mersenne S-box is

$$S_e(x) = x^{2^e-1} + x^{2^n-2},$$

with the exponent vectors $(3, 7, 11)$ for 128s, $(11, 23, 47)$ for 192s, and $(3, 7, 11, 19)$ for 256s. The second term is the formal inverse power, which equals x^{-1} for $x \neq 0$ and 0 for $x = 0$.

AIM3 differs from AIM2 in two places [CHK+26]. The input affine layer, which depends on iv, replaces the addition of fixed constants to the input, and the specification [CHK+26] credits this layer with removing the attack of Biryukov et al. [BFU26]. The enhanced Mersenne S-box replaces the inverse Mersenne S-box of AIM2 and combines a Mersenne power with an inverse term. The MPC checks each S-box through one bilinear relation. For $x \neq 0$ the S-box factors as $S_e(x) = x^{-1}(x^{2^e} + 1)$, which gives

$$x_\ell \, y_\ell = x_\ell^{2^{e_\ell}} + 1, \qquad 0 \leq \ell \leq L,$$

where $y_L := S_{e_L}(x_L) = pt + ct$ completes the output branch. This relation holds only for nonzero S-box inputs, and key generation therefore rejects a key with a zero S-box input.

2.3 Signing in Five Phases

One signing operation runs T repetitions of the MPC in five phases, as in the specification (Figure 1). For a shared value u_ℓ, we write $u_{j,\ell}$ for the share of party j, and $u_\ell = \sum_j u_{j,\ell}$.

1. **Phase 1: committing to the seeds and the execution views.** Each repetition expands a root seed through a Goldreich–Goldwasser–Micali (GGM) tree [GGM86] into N party seeds. A hash of each leaf with the salt, the repetition index, and the party index gives the commitment and the pseudorandom tape of that party. The tape of party j holds a share of pt, shares of $y_0, \ldots, y_{L-1}$, $L + 1$ random shares $a_{j,\ell}$, and a share c_j. Corrections on the last party $N - 1$ make the shares reconstruct the witness and the S-box outputs and make $c = \sum_{\ell=0}^{L} a_\ell \, y_\ell$. Each party then computes its shares of x_ℓ and $z_\ell = x_\ell^{2^{e_\ell}} + 1$ through the affine layers of AIM3.
2. **Phase 2: challenging the multiplication checking protocol.** The hash H_1 absorbs the commitments and corrections of all repetitions in index order, and its output, the first challenge hash h_1, gives the field challenges ϵ_ℓ of every repetition.
3. **Phase 3: committing to the multiplication check results.** Party j simulates the multiplication checking protocol [CHK+26] and computes

$$\alpha_{j,\ell} = a_{j,\ell} + \epsilon_\ell \, x_{j,\ell}, \qquad v_j = c_j + \sum_\ell \epsilon_\ell \, z_{j,\ell} + \sum_\ell \alpha_\ell \, y_{j,\ell},$$

   where $\alpha_\ell = \sum_j \alpha_{j,\ell}$. This simulation is the *check phase*.
4. **Phase 4: challenging the views of the MPC protocol.** The hash H_2 absorbs h_1, the salt, and the α and v shares of all repetitions in index order, and its output, the second challenge hash h_2, selects one unopened party per repetition.
5. **Phase 5: opening the views of the MPC and checking protocols.** The opening adds to the signature an all-but-one tree opening, the commitment of the unopened party, the corrections, and the $L + 1$ missing α shares.

The T repetitions of phase 1 are independent of each other, but H_1 and H_2 absorb the repetitions in index order, and phases 2 to 5 wait for them (Figure 1). On the host, the SHAKE hashes and the linear map of a repetition take the most time, which is consistent with the largest gains of the host code in Subsection 6.3 (Table 7).

Verification has fewer ordering dependencies, because the signature holds h_1 and h_2, and the recomputed H_1 and H_2 can therefore absorb each repetition at its end.

Three maps of phase 1. Phase 1 of repetition r consists of three maps that differ in their cost and in their inputs. The Keccak-dense map K expands the seed tree and derives the commitments and tapes of all parties, and the correction map C computes the difference between the sum of the tape shares and the witness,

$$(\mathsf{com}_r, \mathsf{tape}_r) \leftarrow \mathsf{K}(\mathsf{salt}, \mathsf{sd}_r), \qquad \delta_r \leftarrow \mathsf{C}(\mathsf{tape}_r, pt),$$

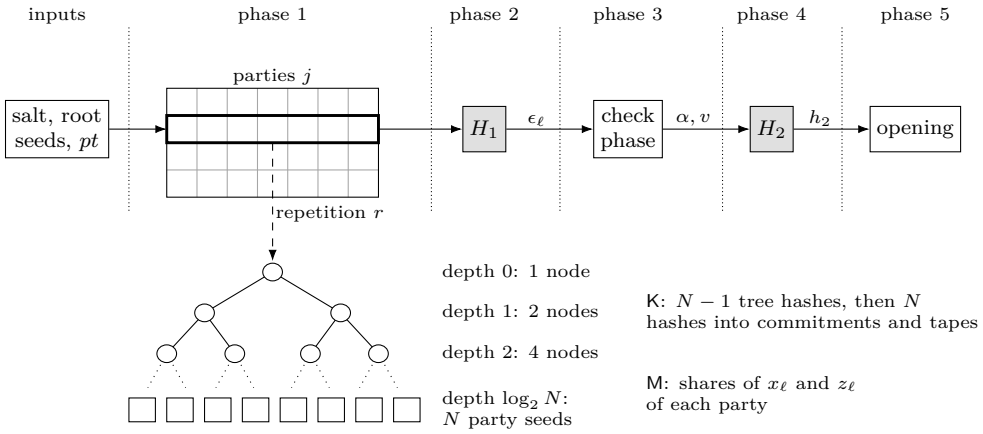

Figure 1: The five phases of one signing operation. Rows of the phase 1 grid are repetitions, and repetition r is expanded into its GGM tree. Shaded boxes absorb the repetitions in index order.

Table 2: Grid axis that each part of the implementation divides or packs into vector elements.

Work	Processor	Axis	Element or bit holds	Section
partition of phase 1	host, NPU	repetitions	—	5
K (tree, leaves)	host	parties	one of 2 Keccak states	3.3
K (tree, leaves)	NPU	parties	one of 16 Keccak states	4.2
M	host	parties	one bit of one share	3.1
M	NPU	repetitions, parties	one bit of one share, 4 repetitions per vector	4.3
H_1, check phase, H_2	host	—	index order, not vectorized	3.3

where sd_r is the root seed of repetition r and $\mathsf{tape}_r = (\mathsf{tape}_{r,0}, \ldots, \mathsf{tape}_{r,N-1})$. The linear map M computes the check operands x_ℓ, z_ℓ of every party from its tape. Because the affine layers of AIM3 act on each share on its own, M decomposes into one map per party,

$$\mathsf{st}_{r,j} \leftarrow \mathsf{M}(\mathsf{tape}_{r,j}) \quad \text{for } j < N-1, \qquad \mathsf{st}_{r,N-1} \leftarrow \mathsf{M}(\mathsf{tape}_{r,N-1} \oplus \delta_r),$$

where $\mathsf{st}_{r,j}$ holds the shares of x_ℓ and z_ℓ of party j. The corrections therefore enter the MPC only through party $N-1$. Only C reads pt, and only M of party $N-1$ reads δ_r.

Work grid and parallel axes. The work of phase 1 forms a grid of T repetitions by N parties (Figure 1). Each repetition expands its GGM tree depth by depth, where each depth holds twice the nodes of the depth above it and needs their outputs, so that the tree takes $N-1$ hashes in $\log_2 N$ sequential steps. Each of the N party seeds then takes one hash into its commitment and tape, and one repetition therefore holds $2N-1$ hashes, or 511 for $N = 256$. Within one repetition, the hashes of one tree depth, the N leaf hashes, and M of the N parties are independent, because M applies the same public matrices to every share. The partition between the CPU and the NPU divides the repetitions. The host kernels fill the elements of their vectors along the parties, and the NPU kernel of M also packs four repetitions into one vector (Table 2).

2.4 Target Platform

The SM8650 (Snapdragon 8 Gen 3) integrates a Qualcomm Kryo CPU and a Qualcomm Hexagon NPU on one SoC [Qua23, Qua26]. The Kryo CPU has one prime, five perfor-

Table 3: Cores of the SM8650 on the TB321FU.

Cores	Microarchitecture	Max. freq. (GHz)	Use
0–1	Cortex-A520	2.266	–
2–4	Cortex-A720	3.149	2C: core 4, 3C: cores 3–4
5–6	Cortex-A720	2.957	–
7	Cortex-X4	3.302	host core of every configuration

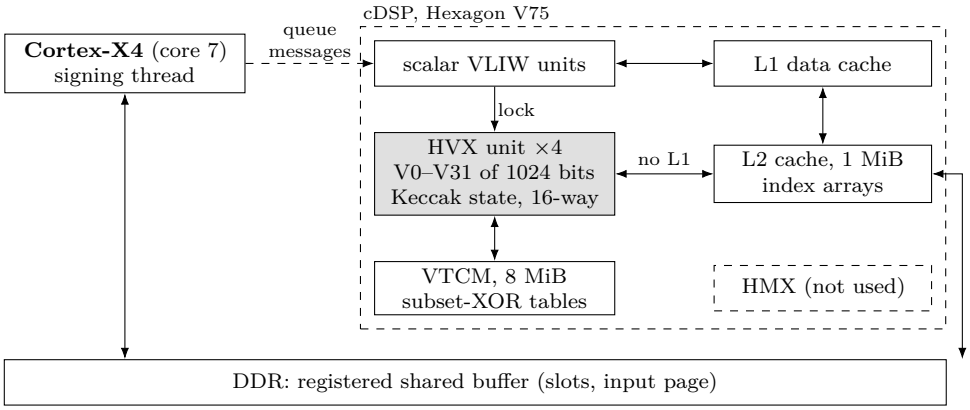

Figure 2: Parts of the SM8650 that the signer uses, with the data in each memory.

mance, and two efficiency cores [Qua23], which the CPU part numbers of the TB321FU tablet identify as one Cortex-X4, five Cortex-A720, and two Cortex-A520 cores (Table 3). A *configuration* is the set of processors that one operation uses. We write 1C, 2C, and 3C for one, two, and three CPU cores, and 1C+NPU and 2C+NPU for one and two cores with the NPU. Our host code uses the 64-bit carry-less multiply PMULL of FEAT_PMULL and the FEAT_SHA3 instructions [Arm26], and the Features field of `/proc/cpuinfo` on the TB321FU lists both. The NPU is one Hexagon V75 core [Qua25], the cDSP, whose scalar VLIW units issue all instructions. A software thread locks one HVX unit through the QuRT kernel of the cDSP [Qua25], and one HVX unit holds thirty-two 1024-bit vector registers [Qua24b]. Our kernels use four HVX units of the one core, and the evaluation therefore counts the NPU as one processor. The Hexagon Matrix Extension (HMX) multiplies low-precision integer and FP16 tiles [HWW+26]. Our kernels do not use HMX, for which the SDK provides intrinsics and lock calls but no instruction manual [Qua25].

The memory of the cDSP has three levels that matter for our kernels. The scalar VLIW units load through an L1 data cache, and the scalar and HVX units share an L2 cache [Qua24b], whose size Hao et al. report as 1 MiB [HWW+26]. HVX loads and stores bypass the L1 data cache and access the L2 cache directly, and a scalar load of data that HVX wrote therefore first fetches the data into the L1 cache [Qua24b]. The cDSP also has vector tightly coupled memory (VTCM), a local memory for scratch buffers [Qua24b], and a query of the cDSP on the TB321FU reports 8 MiB of it (Figure 2). VTCM never evicts its data, and it is faster and uses less power than the L2 cache [Qua24b]. The subset-XOR tables of M take up to 4 MiB for 256s (Subsection 4.3), more than the L2 cache, and every output plane reads them again. Our NPU kernels therefore keep these tables in VTCM, and Subsection 6.4 measures a DDR working space instead. The kernels keep the table indices in cached memory, because scalar byte loads from VTCM made the matrix stage 18 times slower in our test.

Our Hexagon shared object runs in a user protection domain on the cDSP, apart from the Android process on the CPU, and FastRPC marshals the arguments and buffers of each call across this boundary [Qua25, Qua24a]. We register the FastRPC buffers once

and keep one dspqueue, the asynchronous message queue of the Hexagon SDK [Qua25], open for the life of the process. One message of that queue then carries a small descriptor and references to slices of the registered buffers. Each queue message still costs the driver path, the remote scheduling, the synchronization, and the cache maintenance of the buffers that it names, because the caches of the host and the cDSP are not coherent for these buffers [Qua25]. These offload overheads lie on the critical path of the operation, and their size sets how finely Section 5 can divide the work.

The evaluation also uses two retail phones without root. A Galaxy S25 holds the SM8750 (Snapdragon 8 Elite), whose prime cores 6 and 7 reach 4.474 GHz and whose cores 0 to 5 reach 3.533 GHz. Its NPU is a Hexagon V79 cDSP [Qua25] with 8 MiB of VTCM, on which our kernels use six HVX units, and its clock measures 2.112 GHz. A Galaxy S23 holds the SM8550 (Snapdragon 8 Gen 2), whose core 7 reaches at most 2.842 GHz on the cool phone, and whose cores 3 to 6 run at up to 2.592 GHz, below their hardware maximum of 2.803 GHz. Its NPU is a Hexagon V73 cDSP [Qua25] with 8 MiB of VTCM, on which our kernels use four HVX units, and its clock measures 1.478 GHz. A cDSP query gives the VTCM sizes, the HVX unit counts, and the cDSP clocks (1.939 GHz on the SM8650), and sysfs gives the core clocks.

3 AArch64 Implementation

Every configuration runs the host code of this section, and the NPU configurations also run it on the host side. No comparison therefore sets a tuned host against an untuned one. The host code also sets the length of every NPU configuration, because the host keeps the corrections, H_1 and H_2, and the check phase (Section 2.3), and the host path ends the operation (Subsection 5.1). Against the reference implementation (Ref), the section replaces the MPC of phase 1 and the setup (Subsection 3.1), the check phase (Subsection 3.2), and the host SHAKE (Subsection 3.3).

3.1 Bit-Sliced AIM3 MPC

Kwon et al. [KLL+25] group the operations of all N parties into one function call for each field operation, and we apply this grouping to single bits. Let $s_j \in \mathbb{F}_2^n$ be a share of party j, and let the bit plane $P_i = (s_{0,i}, \ldots, s_{N-1,i})$ hold bit i of the shares of all parties in four 64-bit words. Bit o of Λs_j is $\bigoplus_i \Lambda_{o,i}\, s_{j,i}$ for a binary matrix Λ and every party j, and

$$Q_o = \bigoplus_{i:\, \Lambda_{o,i}=1} P_i$$

therefore holds bit o of Λs_j for all 256 parties at once, without any field multiplication.

The Frobenius map $x \mapsto x^{2^e}$ is linear over $\mathbb{F}_2$ with a matrix F_e, and the shares of the S-box inputs and of their powers are therefore affine in pt:

$$x_\ell = A_\ell\, pt + b_\ell, \qquad z_\ell = x_\ell^{2^{e_\ell}} + 1 = F_{e_\ell} A_\ell\, pt + F_{e_\ell} b_\ell + 1, \qquad 0 \le \ell < L.$$

The output branch gives x_L and z_L from $y_0, \ldots, y_{L-1}$ with $A_{L+\ell}$ and $F_{e_L} A_{L+\ell}$. A repetition therefore applies the $4L$ public matrices A_ℓ, $F_{e_\ell} A_\ell$, $A_{L+\ell}$, and $F_{e_L} A_{L+\ell}$, which we compute once per key, to $L+1$ bit-sliced inputs and computes no field square. Additive sharing adds each constant term to one share, and the constants enter the shares of party $N-1$. The reference implementation instead computes each product for each party with n masked XORs of n-bit columns and squares x_ℓ in the field e_ℓ times [Sam26].

Row o of a public matrix Λ selects the input planes whose XOR gives output plane o, which costs up to n plane XORs. We instead divide the n input planes into groups of w planes, with $w = 4$ on the host, and group g receives a subset-XOR table $\mathcal{T}_g$ of 2^w planes,

Algorithm 1 Subset-XOR multiplication of bit planes by a public matrix Λ, with groups of w planes

Input: bit planes $P_0, \ldots, P_{n-1}$, the w-bit chunks $\Lambda_{o,g}$ of the rows of Λ
Output: output planes $\mathrm{out}_0, \ldots, \mathrm{out}_{n-1}$

for $g = 0$ **to** $n/w - 1$ **do** ▷ tables, once per input
  $r \leftarrow 0$, $\mathcal{T}_g[0] \leftarrow 0$
  for $i = 1$ **to** $2^w - 1$ **do**
    $r \leftarrow r \oplus P_{gw+\mathrm{ctz}(i)}$, $\mathcal{T}_g[i \oplus \lfloor i/2 \rfloor] \leftarrow r$ ▷ Gray-code order
for $o = 0$ **to** $n - 1$ **do** ▷ once per matrix
  $\mathrm{out}_o \leftarrow \bigoplus_{g=0}^{n/w-1} \mathcal{T}_g[\Lambda_{o,g}]$ ▷ index from Λ only

whose entry s is the XOR of the planes that the w bits of s select. Output plane o is then the XOR of the entries $\mathcal{T}_g[\Lambda_{o,g}]$, where $\Lambda_{o,g}$ is the w-bit chunk of row o at group g (Algorithm 1), and one output plane therefore costs n/w table reads instead of about $n/2$ plane XORs for a random row. Subset-XOR multiplication is the bit-sliced form of the method of the four Russians for multiplication (M4RM) [ABH10], which derives from the algorithm of Arlazarov et al. [ADKF70]. The table index is always a chunk of the public matrix, and every memory address is therefore independent of the secret. Algorithm 1 builds each table in Gray-code order with one XOR and one store per entry, and the NPU kernel of Subsection 4.3 uses the same tables with $w = 8$ for 192s and 256s.

The tables of one input take $16 \cdot n/4$ planes, at most 32 KiB for 256s, which fits in the L1 data cache, one 64-bit load gives the indices of eight groups, and a repetition performs $4L \cdot n \cdot n/4 = Ln^2$ table reads of 32 bytes each. One matrix therefore costs $n^2/256$ 64-bit words per party, against $n^2/64$ words per party in the reference implementation, and the tables of an input, $4n$ plane XORs, serve all matrices that read the input. Two 64×64 bit transposes with the NEON TRN1 and TRN2 instructions in 32-, 16-, and 8-bit stages convert between the per-party order of the tapes and the bit planes. The MPC writes the inputs of the check phase, the shares of x, z, a, y, and c, back in the per-party order. In a microbenchmark on the X4 (core 7, minimum of three runs), one repetition of the MPC, transposes included, takes 16.5, 37.1, and 96.9 µs for 128s, 192s, and 256s. Table 7 measures these techniques together with the rest of this section.

Setup. The linear layer, its folded matrices, and the index arrays of the subset-XOR multiplication depend only on iv, and we compute them once per key. The cache key compares b and the first and last row of every matrix, which a different iv changes except with negligible probability. The setup of an operation computes the S-box outputs y_ℓ from pt with one field inversion each. Instead of the Fermat chain of $n-1$ multiplications of Ref, we use the Itoh–Tsujii chain [IT88]. With $\beta_k = x^{2^k-1}$, the chain follows an addition chain of $n-1$ with $\beta_{i+j} = \beta_i^{2^j}\beta_j$ and returns $x^{-1} = \beta_{n-1}^2$, which needs $\lfloor \log_2(n-1) \rfloor + \mathrm{wt}(n-1) - 1$ multiplications (12 for 128s, 13 for 192s, and 14 for 256s) and $n-1$ squarings with the PMULL multiply and square of Kwon et al. Its sequence of operations depends only on n, and the inversion is therefore constant time with respect to pt.

3.2 Check Phase with Lazy Reduction

The check phase evaluates the α and v equations of Subsection 2.3 for all parties, and the reference implementation reduces each of its products after a portable carry-less multiply [Sam26]. We keep the lazy reduction of Kwon et al. [KLL+25] and compute directly from the check-phase inputs in NEON registers. A field element of W 64-bit words occupies $\lceil W/2 \rceil$ 128-bit chunks. The product of the words a_i and b_k lies at bit offset $64(i+k)$. PMULL multiplies a 128-bit chunk of a with a chunk of b and with its half-swapped copy,

and the products with $i+k$ even XOR into the accumulator E and those with $i+k$ odd into O. One 64-bit lane shift of O merges both accumulators at the end, and all products of one output stay unreduced until then. For a 192s element of two 128-bit chunks, the generic product computes 2×2 chunk products of four PMULLs each, and a dedicated product of the three words uses 9 PMULLs instead of these 16. The share v_j sums $2(L+1)$ products, and the lazy reduction therefore reduces once instead of $2(L+1)$ times. The field polynomial is $f = x^n + \varphi(x)$ with $\deg \varphi < 64$, where φ is $x^7 + x^2 + x + 1$ for 128s and 192s and $x^{10} + x^5 + x^2 + 1$ for 256s, and $x^n \equiv \varphi(x)$ reduces an unreduced product $lo + hi \cdot x^n$ to $lo + hi \cdot \varphi(x)$ modulo f. We multiply the W words of hi by φ with W PMULLs and fold the part above x^n, whose degree is below 20, with one more PMULL. The reduction therefore costs $W+1$ PMULLs in NEON registers. The same microbenchmark gives 4.3, 7.7, and 13.0 µs per repetition of the check phase for 128s, 192s, and 256s.

3.3 Host Keccak and the Hashes H_1 and H_2

The host computes the map K of every repetition that the NPU does not take and the hashes H_1 and H_2, which lie on the host path that ends the operation (Subsection 5.1). The tree, commitment, and tape hashes of a repetition are many short SHAKE instances without dependencies between them, which the reference implementation computes one at a time with a portable Keccak [Sam26]. We instead compute two instances at a time, as Picnic does with a four-way AVX2 Keccak [KZ20] and Kwon et al. with a two-way NEON Keccak [KLL+25]. Our host uses the two-way NEON Keccak-f[1600] of mlkem-native [The26, BK22], whose permutation uses the FEAT_SHA3 instructions EOR3, RAX1, XAR, and BCAX. EOR3 computes the three-input XORs of the column parities of θ, RAX1 computes $D[x] = C[x-1] \oplus \mathrm{rot}(C[x+1], 1)$ of θ, XAR merges the θ XOR with the rotation of ρ, and BCAX computes the step $a \oplus (\neg b \wedge c)$ of χ. The host hashes the nodes of one tree depth and the 256 leaves in groups of four with two calls of the two-way permutation, and only depths with fewer than four nodes leave positions unused. The single sponges H_1 and H_2 use the scalar Keccak-f[1600] of the same library, and our sponge code absorbs and squeezes whole 64-bit words instead of bytes. In Table 7, the mlkem-native Keccak and the word-wise sponge give a factor of 1.6 to 2.0 after the grouped MPC. The signing thread absorbs each repetition into H_1 as soon as its phase 1 ends and all earlier repetitions are absorbed, and no separate pass over the commitments remains before the check phase. The hash H_2 absorbs the α and v shares in index order, and Subsection 5.2 moves part of that absorb into the NPU wait.

Early H_2 absorb on several cores. A direct division of the check phase over several cores starts H_2 only after every core ends. The first core instead absorbs its repetitions into H_2 while the other cores compute the rest. In 12 alternating runs of both divisions, the early absorb takes 0.971 to 0.976 of the time of the direct division in 2C+NPU and 0.969 to 0.983 in 3C, and in 2C it gains only for 128s, with 0.988.

4 HVX Implementation on the Hexagon NPU

The NPU side runs as one persistent service in the cDSP user domain and computes the two maps of phase 1 that do not read the secret key (Section 2.3). A *SHAKE packet* computes K for a range of repetitions, and one *MPC packet* per operation computes M for a set of repetitions. We write S for the repetitions of the SHAKE packets and $D \subseteq S$ for the repetitions of the MPC packet, and Section 5 selects both sets. The service keeps the offload overhead of Subsection 2.4 low with one shared buffer (Subsection 4.1). The service computes K with a 16-way packed Keccak (Subsection 4.2), computes M with a

bit-sliced subset-XOR multiplication over four repetitions at a time (Subsection 4.3), and divides the work over the HVX units, four on the SM8650, with atomic counters.

4.1 FastRPC Queue and Shared Slots

The host registers one shared buffer when the process starts, and every queue message of an operation names slices of that buffer instead of copying data. The buffer holds one slot per repetition, in which the NPU writes the commitments and tapes of all N parties and the host reads them in place. An input page in the same buffer holds the per-operation inputs, such as the salt and the root seeds of the NPU repetitions. The first SHAKE packet of an operation names the input page, so that no separate FastRPC call precedes it, and the index arrays of the subset-XOR multiplication move to the NPU once per key. A queue message that does no work returns to the host after 34 µs on the SM8650. In bursts of such messages, each further message adds 8 µs without a buffer and 19 to 22 µs with a slice of 256 KiB to 1 MiB. This per-packet cost approximates the overhead β of one packet in Subsection 5.1, and in the trace of that section the gaps between the packets of the 192s and 256s operations exceed it by up to 18 µs.

4.2 16-Way Packed HVX Keccak

The seed tree and the commitments of one repetition are independent SHAKE instances over the parties, and HVX has no 64-bit element operations [Qua24b]. Caullery stores one Keccak state in five HVX vectors of five 64-bit words each, and one permutation takes 6,260 cycles on the Hexagon V79 simulator [Cau25]. Our kernel instead packs 16 states into 25 vectors, and on the same simulator one call permutes the 16 states in 4,270 cycles, or in 3,379 cycles as the mean of 512 consecutive calls. The kernel also runs on the V75 device inside the signing operation. Each Keccak lane occupies one 1024-bit vector, whose first half holds the low 32-bit words of the 16 states and whose second half holds the high words, and the 25 state vectors stay in the 32 HVX registers during all 24 rounds. Packing and unpacking convert between this layout and the instance-major order of the sponge inputs and outputs with `vmux` and `vror`. A layout with the low and high words of each lane in two vectors holds 32 states, but its 50 vectors exceed the 32 registers of one HVX unit. Because every round reads all 25 lanes, that layout spills state to memory in every round.

In the packed layout, a 64-bit rotation of every lane by $0 < r < 32$ bits is

$$\mathrm{rot}(V, r) = (V \ll r) \vee \big(\mathrm{swap}(V) \gg (32 - r)\big),$$

where both shifts act on each 32-bit word, and $\mathrm{swap}(V)$, one `vror` by half the vector, exchanges the low and the high words of every lane. A rotation by $r \geq 32$ applies swap first and rotates by $r - 32$. In HVX assembly, a rotation by $r < 32$ is

```
V1 = vror(V0,R64) // swap(V)
V2.uw = vlsr(V1.uw,R32mr) // swap(V) >> (32-r)
V2.w += vasl(V0.w,Rr) // + (V << r)
```

where the scalar registers hold 64, $32 - r$, and r, and the addition of `vaslacc` gives the OR, because the two terms have no common bit. One rotation therefore costs one `vror` and two shifts, and a rotation by one replaces its left shift with the addition $V + V$, so that the 29 rotations of θ and ρ take 52 shifts instead of 58 (Table 4). The step χ uses the lane-complementing transform [BDP+12], which stores six lanes complemented and needs six `vnot` per round. A VLIW instruction packet, the group of instructions that the core issues together and not a queue message, holds at most one shift [Qua24b], and the 52 shifts of a round therefore need at least 52 instruction packets. Our assembly unrolls twelve rounds per loop iteration and takes 65.8 packets per round.

Table 4: HVX instructions in one round of the 16-way packed Keccak. Shifts are `vlsr` and `vaslacc`, and other is `vadd`, `vand`, `vor`, `vnot`, and one load.

Step	`vxor`	other	`vror`	shifts	total
θ	50	10	5	5	70
ρ, π	0	2	24	47	73
χ	25	31	0	0	56
ι	1	1	0	0	2
round	76	44	29	52	201

The inputs of one task have the salt and the repetition index in common, and the kernel therefore splats each common word into its lane and places only the $B/4+1$ words that differ, where B is the seed length in bytes. The output of each instance is a run of at most 128 bytes, which the kernel writes with one `vror` and at most two aligned stores under byte masks. A SHAKE packet divides the grid rows of its repetitions into two kinds of tasks. A tree task expands the seed tree of one repetition depth by depth. Each operation of the kernel packs up to 32 nodes of one depth into two packed states of 16 instances each, and the depths with fewer than 16 nodes therefore leave instance positions unused (Section 2.3). A commitment task hashes 32 parties of one repetition into their commitments and tapes, and it follows the tree task of its repetition. The HVX units take the next task from an atomic counter, so that every packet uses all units.

Cost of a 16-way hash. A 16-way hash processes up to 16 instances in one packed state. We time the deployed 16-way hashes in a benchmark build on the SM8650, with dummy inputs, core 7 at 3.302 GHz, and the median of 15 timings. When four HVX units run it at the same time, a 16-way hash costs almost as much with one instance as with 16 instances, 0.82 to 0.88 of the time, because the permutation and the unpacking do not depend on the number of instances. The four top depths of a seed tree therefore cost about four full 16-way hashes, although they hold only 15 hashes. These 15 hashes are 15 of the 511 hashes of one repetition in the s sets, but 15 of 31 in the f sets (Subsection 2.1). At full width, one HVX unit takes about twice the time per hash of one X4 core, 0.17 against 0.084 µs at the 128-bit level. The same map K of one s-set repetition therefore takes 1.77 to 1.94 times less time on the four units together, in a packet of five repetitions, than on one core. The four units take 25.1, 35.0, and 45.3 µs for 128s, 192s, and 256s, and the core takes 44.4, 65.8, and 88.1 µs. These two times are s_N and s_H of Subsection 5.1.

4.3 Bit-Sliced Linear Map on HVX

The MPC packet computes the map M (Section 2.3) for the repetitions of D. All repetitions use the same public matrices, and a 1024-bit vector can therefore hold one bit plane of four repetitions, 32 bytes each, so that one read of a subset-XOR table serves all four. The MPC packet processes the repetitions of D in groups of four, and each group passes the six stages of Table 5. Because the stages depend on each other, the packet lists the work of all groups as numbered work items in stage order. Each HVX unit takes the next item from an atomic counter and waits until the stages that the item reads are complete, and therefore waits only for items with smaller numbers.

The DSP microbenchmark runs the MPC packet alone on the cDSP with random tapes and checks its outputs against the host MPC. The matrix stage takes the most time at every parameter set (Table 5), and the profile suggests that its table reads limit it. For 192s and 256s, the tables therefore use groups of $w = 8$ planes with 256 entries each, which halves the reads per output plane from $n/4$ to $n/8$, and groups of eight planes take

Table 5: Busy time in µs per stage of the MPC packet for two groups of four repetitions, mean over four HVX units at 1.939 GHz. The 192s tables row is from a separate profile.

Stage	Work	128s	192s	256s
input	tape words of 32 parties to bit planes	16.2	19.7	29.8
word transpose	planes of four repetitions into one vector	1.0	1.4	2.8
tables	subset-XOR tables of each input	1.0	14.4	25.5
matrix	output planes from table reads	27.7	32.5	82.3
output	planes back to per-party order	15.2	22.8	40.5
write	x_ℓ, z_ℓ and the other check-phase inputs	6.7	15.6	13.8

0.887 and 0.700 of the elapsed time of groups of four planes. The 128s set keeps groups of four planes, because groups of eight take 0.993 of its 1C+NPU signing time. For 256s, the shared tables of one MPC packet take 4 MiB, which stay in VTCM (Subsection 2.4). Each bit plane holds all N parties, and the MPC packet therefore also computes party $N-1$, from its tape before the correction, because the host computes the corrections only later. The host replaces that output with check-phase inputs from the corrected tape.

5 CPU–NPU Work Partitioning

In this section, we derive the partition of one signing operation from two properties of the operation. The dependency graph of a repetition (Section 2.3) shows which work the NPU can compute without the secret key, and the critical path across the two processors, with the costs of Section 3 and Section 4, shows which part of that work shortens the operation. The selected partition gives the NPU the map K of the last repetitions and, for the last d of them, the map M of all parties but the last.

The phases and maps of Section 2 give two rules for any partition. Phase 1 can move to another processor repetition by repetition, but H_1, H_2, and the check phase stay on the processor that receives the phase 1 outputs in index order. The map K and the map M of the parties $0, \ldots, N-2$ need no access to pt or δ_r, but C and the map M of party $N-1$ read them and stay on the host. The critical path of Subsection 5.1 then selects the part of the work without the secret key that shortens the operation.

5.1 Critical Path of a Partition

Three values describe a partition. The value k divides the repetitions into the host repetitions $P = [0, k)$ and the NPU repetitions $S = [k, T)$, S is divided into G SHAKE packets

$$R_g = \big[\, k + \lfloor g\tfrac{T-k}{G} + \tfrac{1}{2} \rfloor,\ k + \lfloor (g{+}1)\tfrac{T-k}{G} + \tfrac{1}{2} \rfloor \big), \qquad g \in [0, G),$$

and $D = [T-d, T) \subseteq S$ holds the last d repetitions, whose linear map the NPU also computes. The NPU computes K for S and M for D, of which the host keeps the parties $0, \ldots, N-2$. The host computes K for P, C for all repetitions, M outside D and for party $N-1$ of D, and H_1, check phase, H_2, and opening (Algorithm 2, example in Figure 3).

The two processors work in parallel until the host consumes the outputs of the NPU in index order, and the operation therefore ends with the later of two paths. Let s_N and m_N be the NPU time per repetition for K and M (Subsection 4.2 and Subsection 4.3), and β the overhead of one packet (Subsection 4.1). Let s_H, m_H, and c_H be the host time per repetition for K, for M, and to consume one repetition of a packet, that is, to compute C and absorb the repetition into H_1 (lines 4 to 6 of Algorithm 2). The NPU path and the host path before the check phase, without waits, end at

$$E_N = (G{+}1)\,\beta + (T-k)\,s_N + d\,m_N, \qquad E_H = k\,(s_H + m_H) + (T-k)\,c_H + (T-k-d)\,m_H,$$

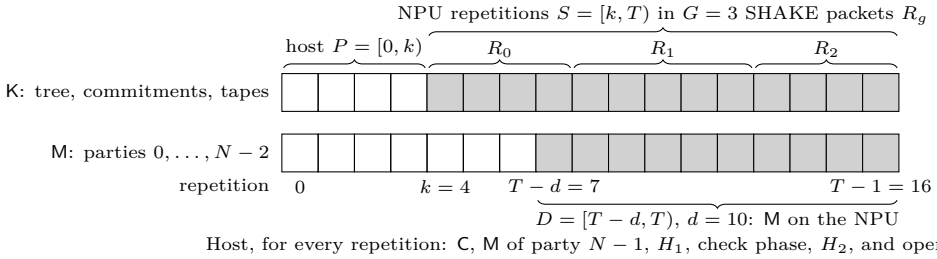

Figure 3: Partition of 1C+NPU for 128s on the SM8650 at $(k, G, d) = (4, 3, 10)$. Shaded cells run on the NPU.

Algorithm 2 Host side of one partitioned signing operation

Input: message msg, secret key sk, partition (k, G, d)
Output: signature σ

1: Derive the salt and the root seeds, and write the input page (Subsection 4.1)
2: SEND SHAKE packets $R_0, \ldots, R_{G-1}$ and the MPC packet for D ▷ the NPU runs them in order
3: Compute K, C, M for P, and absorb P into H_1
4: **for** $g = 0$ **to** $G - 1$ **do** ▷ consume R_g
5: WAIT for R_g, and compute C for its repetitions
6: Compute M for $R_g \setminus D$, and absorb R_g into H_1
7: $(\epsilon_0, \ldots, \epsilon_L) \leftarrow \text{EXPAND}(h_1)$ ▷ host path ends, E_H
8: Run the check phase of $[0, T - d)$, and absorb it into H_2
9: WAIT for the MPC packet, recompute M of party $N - 1$ of D ▷ NPU path ends, E_N
10: Run the check phase of D, and absorb it into H_2
11: $\sigma \leftarrow \text{OPEN}(h_2)$ ▷ unopened parties and opening

and the operation ends at about $\max(E_H, E_N)$ plus the part of the check phase and H_2 that the MPC packet cannot overlap. A smaller k lowers E_H by $s_H - c_H$ and raises E_N by s_N, a larger d removes m_H from E_H and adds m_N to E_N, and a larger G adds β per packet to E_N. The model omits the benefit of a larger G, an earlier start of the consume on the host. An improvement of an NPU kernel shortens the operation only while E_N is later than E_H. If E_H is later than E_N, only work that leaves the host path, through a smaller k or a larger d, shortens the operation. We use the model for the direction of each value, and a local search on the device selects the three values (Subsection 6.2).

We call the partition with $d = 0$ the *Keccak-only offload* and a partition with $d > 0$ a *linear-map offload*. The Keccak-only offload is the direct offload of complete repetitions under the condition that pt stays on the host, and its E_H keeps the term $(T - k)\, m_H$. A trace of the Keccak-only offload shows that E_H is later than E_N, and a faster NPU kernel therefore cannot shorten the operation, as the model predicts (Figure 4). A partition must instead move work from the end of the host path to the NPU without moving the secret-dependent map C.

A build of our code that also records host and NPU events on the 19.2 MHz system counter of both sides places the two sides of one operation on one time axis (Figure 4). The trace series runs on battery power, with the Keccak-only offload at the (k, G) of a separate search with $d = 0$. In the Keccak-only offload, the NPU work ends at E_N of 421 µs for 128s, 1,015 µs for 192s, and 2,338 µs for 256s (dashed lines in Figure 4), and the host needs another 557 µs, 1,467 µs, and 3,921 µs (braces). The host computes C and M of the NPU repetitions for 206 µs, 551 µs, and 1,860 µs of that time. The NPU is idle for 63 % of the operation for 128s, 64 % for 192s, and 67 % for 256s.

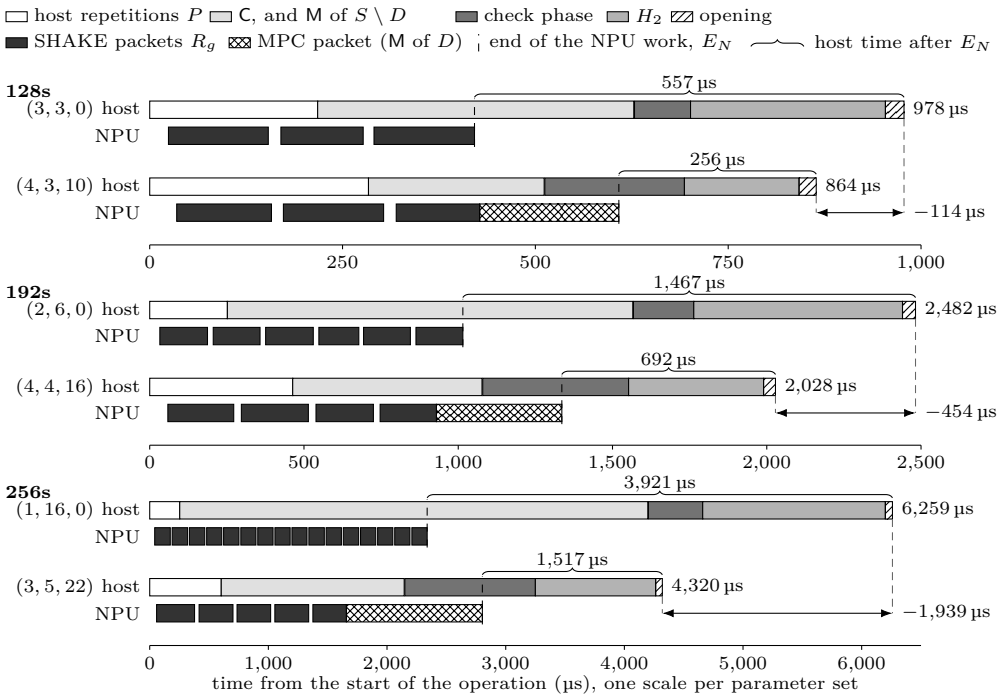

Figure 4: Mean timeline of traced 1C+NPU signing operations for the Keccak-only and linear-map offloads. Arrows mark the time that the linear-map offload saves.

5.2 Schedule of a Linear-Map Offload

The linear-map offload overlaps the two processors in three places (Algorithm 2). The host computes P while the NPU computes R_0, and it then consumes each packet R_g, applies its corrections, and absorbs it into H_1 while the NPU computes R_{g+1}. After the last SHAKE packet the NPU runs the MPC packet for D, and during that packet the host runs the check phase of the repetitions before D and absorbs them into H_2. The MPC packet is separate from the SHAKE packets, because the kernel of Subsection 4.3 can start a group of four repetitions only after K is complete for all four repetitions. In 2C+NPU, the second core also takes part of P, consumes part of the packets, and takes part of the check phase, which the two cores divide only after the MPC packet. A separate trace series of 2C+NPU on battery power, with 12 to 14 of 15 traced operations per parameter set at 3.302 GHz, shows this order. The MPC packet ends at 403, 947, and 2,176 µs for 128s, 192s, and 256s, and the check phase starts at 508, 1,215, and 2,305 µs. The part of H_2 after the check phase takes 149, 383, and 817 µs (1C+NPU: 149, 438, and 1,013 µs).

The set D holds the last d repetitions of S, because only the end of the host path decides the length of the operation. The host computes M of the first packets while the NPU is still busy, and it is later than the NPU only at the end of the operation, where the part of M that the NPU computes shortens the host path. In the trace of Figure 4, the MPC packet occupies the NPU after its last SHAKE packet, and the NPU is idle for 37 % of the operation for 128s, 41 % for 192s, and 40 % for 256s. The host time after E_N falls from 557 to 256 µs for 128s, from 1,467 to 692 µs for 192s, and from 3,921 to 1,517 µs for 256s. The operation ends 114, 454, and 1,939 µs earlier (arrows), less than this drop, because the MPC packet makes the NPU work end 187, 321, and 465 µs later.

Trust boundary. The partition keeps C and party $N-1$ on the host, but it does not hide the witness from the NPU. The root seeds of S determine every tape of those repetitions

Table 6: Data at the host–NPU boundary for one signing operation. The NPU writes the entries marked †.

NPU receives or writes	Only the host computes
salt, root seeds of S	witness pt
index arrays of the public linear layer	corrections δ_r of all repetitions
commitments and tapes of $S^{\dagger}$	check-phase inputs of party $N-1$
MPC outputs of $D^{\dagger}$, party $N-1$ from its uncorrected tape	H_1, check phase, H_2, opening

and the signature publishes the corrections, and a processor that holds both can therefore recover pt. The NPU therefore belongs to the trusted computing base of the signer. The NPU receives no secret-dependent value that the root seeds do not determine (Table 6).

Verification on the CPU and the NPU. Verification uses the same linear-map offload on the dependency graph of Subsection 2.3, and the host absorbs each repetition into H_1 and H_2 as soon as the repetition ends. On the NPU, the SHAKE packets rebuild the seed trees of S from the opening and compute the tapes and commitments of the $N-1$ opened parties, with a dummy tape in the slot of the unopened party. The MPC packet runs the kernel of Subsection 4.3 without change for D. The host recomputes the MPC outside D and the α and v shares of all repetitions. The host takes the α share of the unopened party from the proof and derives its v share from the other parties, as Ref does [Sam26].

6 Experimental Evaluation

This section measures whether the linear-map offload of Section 5 makes one signing operation and one verification operation faster than a CPU configuration with one more core, and what the shorter operation costs in energy. On all three SoCs, the NPU configurations sign faster than the CPU configurations with one more core, and on the SM8650 no NPU configuration uses measurably more energy per signing operation.

6.1 Experimental Setup

The SM8650 series run on the TB321FU tablet on USB power, with the battery in the state Not charging, the performance governor on all eight cores, no clock cap, and core 7 at its maximum of 3.302 GHz before each process. Each process also waits until every CPU thermal zone is below 60 °C and the battery is below 45 °C. The phones run on USB power with their default governor, because we lack root access, and spinning threads outside every measured set keep the governor from lowering the clock of core 7. The host core is core 7 on every SoC, and on the phones 2C adds core 6 and 3C adds cores 5 and 6.

In the NPU configurations, FastRPC runs in its poll mode, in which the host CPU polls for the cDSP response for up to 3,000 µs, the value that we choose. The poll mode sets a power-management quality-of-service (PM QoS) request of 100 µs, which keeps the host CPU out of its deeper idle states [Qua25]. Each measured process is persistent and signs or verifies 100 of 550 test vectors after 10 warm-up operations. Each test vector holds a key pair, a message, and the signature of Ref. The *settings* of a configuration are the values (k, G, d) of an NPU configuration or the repetition boundaries of the CPU cores. The searches for the settings (Subsection 6.2) use a separate set of test vectors, so that no reported time uses an input of a search. We measure 128s, 192s, and 256s in separate series. In one series, each of 24 *trials* runs every configuration once, as one process at its settings, in an order that shifts cyclically by one position from trial to trial. A table cell

is the median over the trials of the wall-clock median of each process. A comparison of two configurations reports the geometric mean of their per-trial time ratios, with a 95 % bootstrap interval. The phones report only the frequency that the governor requests, and a comparison on a phone therefore keeps a trial only if the median requested frequencies of its two configurations differ by at most 5 %. Because the conditions differ, absolute times of the SM8650 and of the phones are not comparable.

Every timed signing operation compares its signature byte for byte with the output of Ref, every timed verification operation must accept the signature of Ref, and no operation of the time or energy measurements fails this check. The implementation is designed to run in constant time with respect to pt, because the table indices are chunks of the public matrices and the inversion depends only on n (Subsection 3.1). A test vector leakage assessment (TVLA) [GJJR11] of the timing of 1C+NPU on the SM8650 compares a fixed key with random keys of the same iv by the Welch t-test of dudect [RBV17]. The largest $|t|$ is 1.368 with 1,977 traces per class, below the TVLA threshold of 4.5. The TVLA run uses a build that differs from the evaluated build only in the order of the NPU initialization. A memcheck run in the style of ctgrind [Lan10] and TIMECOP [Nei20] finds no branch and no memory address of the host signer that depends on pt, for 1C at the three parameter sets and 2C at 128s and 256s with the portable Keccak. Neither check is a proof, and the memcheck run does not cover the NPU code.

6.2 Search for the Settings

Each configuration runs with the settings that a search selects on its own test vectors, and the measurements contain no process of the search. The file `SETTINGS.md` of the code repository lists the selected settings. A local search measures the neighbours of the current settings, $k \pm 1$, $G \pm 1$, and $d \pm 2$ for the NPU configurations and the repetition boundaries for 2C and 3C. The search moves only to a neighbour that is at least 0.5 % faster in the median of two search runs on the SM8650 and of three on the phones. A coarse grid search confirms the 1C+NPU signing settings of the SM8650 and the SM8750, and on the SM8550 a local search from a faster grid setting gives five of six signing settings. 1C+NPU signing keeps two to four host repetitions on every SoC, and d grows with T.

6.3 Latency

With the NPU counted as one processor, the NPU configurations sign faster than the CPU configurations with one more core on all three SoCs (Table 8). 1C+NPU takes 0.705 to 0.768 of the 2C time on the SM8650, 0.742 to 0.783 on the SM8750, and 0.550 to 0.574 on the SM8550. 2C+NPU takes 0.810 to 0.875 of the 3C time on the SM8650, 0.796 to 0.907 on the SM8750, and 0.672 to 0.698 on the SM8550. On the SM8650, their 95 % intervals are at most 0.007 wide and describe one series, not the variation between series. Both ratios are highest, and the gains therefore smallest, for 128s on the SM8650. This order is consistent with the model of Subsection 5.1, because the work that the NPU takes from the host grows with T and d, but the overhead β of a packet does not. Although the measurement conditions of the SoCs differ, the gain is largest on the SoC with the slowest 1C and smallest on the SoC with the fastest 1C, which is consistent with a host path that ends the operation. 1C+NPU takes 0.507 to 0.574 of the 1C time on the SM8650, 0.562 to 0.642 on the SM8750, and 0.359 to 0.371 on the SM8550.

On the SM8650, the host code signs 4.5 to 6.1 times and verifies 4.4 to 5.9 times faster than Ref (Table 7). Table 7 adds the techniques of Section 3 one at a time, in one series of 24 trials on USB power with the battery Charging or Not charging. The grouped MPC of Kwon et al. [KLL+25] gives a factor of 1.6 to 2.3 over Ref, the mlkem-native Keccak with the word-wise sponge gives a further 1.6 to 2.0, and the other techniques give the last 1.40 to 1.53.

Table 7: Wall-clock medians in ms of 1C on the SM8650 over 24 trials.

	Signing			Verification		
Host code	128s	192s	256s	128s	192s	256s
Ref	7.040	24.369	43.134	6.967	23.896	42.028
+ grouped MPC	4.502	10.652	21.878	4.450	10.599	21.712
+ mlkem-native Keccak, word-wise sponge	2.244	6.098	13.467	2.221	6.088	13.203
+ Section 3	**1.556**	**3.996**	**9.128**	**1.591**	**4.056**	**9.351**

Table 8: Wall-clock medians in ms over 24 trials of 100 operations. The Ref rows come from separate series.

		Signing			Verification		
SoC	Configuration	128s	192s	256s	128s	192s	256s
SM8650	1C	1.553	3.910	8.889	1.597	3.980	9.116
	2C	1.159	2.884	6.402	1.145	2.809	6.216
	3C	0.962	2.438	5.149	0.925	2.299	5.078
	1C+NPU	**0.889**	**2.036**	**4.512**	**0.950**	**2.254**	**4.662**
	2C+NPU	0.845	2.061	4.170	0.918	2.101	4.354
	Ref	7.040	24.369	43.134	6.967	23.896	42.028
SM8750	1C	1.474	3.734	8.639	1.499	3.838	8.838
	2C	1.203	2.946	6.519	1.081	2.647	6.689
	3C	1.137	2.725	5.959	1.080	2.455	5.559
	1C+NPU	**0.946**	**2.245**	**4.851**	**0.999**	**2.454**	**5.344**
	2C+NPU	1.034	2.468	4.729	1.102	2.555	5.002
	Ref	7.418	24.943	46.871	6.462	22.510	40.400
SM8550	1C	3.664	8.663	17.516	3.653	8.650	17.605
	2C	2.365	5.660	11.496	2.256	5.235	11.003
	3C	1.941	4.364	9.070	1.803	4.353	8.919
	1C+NPU	**1.357**	**3.112**	**6.317**	**1.377**	**3.167**	**6.674**
	2C+NPU	1.304	2.944	6.148	1.382	3.071	6.393
	Ref	10.326	33.906	61.671	8.427	29.514	52.839

Against the AIM3 implementations of Lee et al. [LEK+26], our host code gains more over Ref than their AVX2 code for all three parameter sets, and more than their AVX-512 code only for 192s (Table 9). Kwon et al. [KLL+25] and Lee et al. [LES26] implement AIM2, whose one-way function differs, and Table 9 therefore omits them.

The host code also signs faster than Ref on the phones, by 2.8 to 6.5 times in the per-trial ratios of each phone series (Table 8). Table 8 takes its Ref rows from these separate Ref series, and the 1C of each Ref series agrees with the 1C of Table 8 within 3.9 %.

A second host core does not shorten every operation. The second core makes 1C+NPU faster on the SM8650 for 128s and 256s, with ratios of 2C+NPU to 1C+NPU of 0.948 and 0.924, but not for 192s, with 1.012. On the SM8750, 2C+NPU signs 128s and 192s more slowly than 1C+NPU (1.088 and 1.101), and a grid search over the settings gives the same result. On this SoC, 1C+NPU is therefore the faster NPU configuration for these two sets, and we leave the cause of the slowdown to future work. On the SM8550, the ratio is 0.961 for 128s and 256s, and the interval of 192s, 0.946 to 1.037, includes 1.

At 256s, the NPU configurations of the SM8550 show a rare slow mode, about 38 % slower at the same requested frequency, in one of the 24 trials of 1C+NPU signing and of 2C+NPU verification. The 5 % frequency rule keeps 20 to 24 of 24 trials per comparison on the SM8750 and all 24 on the SM8550. The slow trial therefore stays in its comparisons,

Table 9: Signing time in ms of AIM3 implementations, with the factor over Ref on the same processor in parentheses. We convert the cycles of Lee et al. at their fixed clocks.

Implementation	Processor	128s	192s	256s
AVX2 [LEK+26]	Core i7-1165G7, 2.8 GHz	5.70 (3.2)	14.12 (3.9)	26.29 (4.3)
AVX-512 [LEK+26]	Core i7-1165G7, 2.8 GHz	3.88 (4.7)	9.82 (5.6)	17.70 (6.4)
MVE [LEK+26]	Cortex-M55, 600 MHz	1,126 (1.3)	2,790 (1.5)	6,366 (1.9)
host code, 1C	Cortex-X4, 3.302 GHz	1.556 (4.5)	3.996 (6.1)	9.128 (4.7)
1C+NPU	Cortex-X4 and cDSP	0.889 (7.9)	2.036 (12.0)	4.512 (9.6)

where it moves the geometric mean over 24 trials by about 1.4 %. We leave the cause of the slow mode to future work.

The same linear-map offload also accelerates verification, but with smaller margins (Table 8). 1C+NPU takes 0.750 to 0.832 of the 2C time on the SM8650, 0.799 to 0.928 on the SM8750, and 0.605 to 0.612 on the SM8550. On the SM8750, 2C+NPU takes 1.020 and 1.040 of the 3C time for 128s and 192s, and the NPU therefore does not replace a third core there for these sets. The smaller margins are consistent with the fewer ordering dependencies of verification (Subsection 2.3), in which a CPU configuration also absorbs every repetition into H_1 and H_2 as soon as the repetition ends.

Against the Keccak-only offload, the linear-map offload gives a gain that grows with the parameter set, which is consistent with the part of M that the NPU removes from the end of the host path (d grows with T, Subsection 6.2). In a separate series of 12 trials on battery power, the Keccak-only offload with its own settings (Subsection 5.1) takes 0.837, 0.857, and 0.969 of the 2C time, and 1C+NPU takes 0.895, 0.806, and 0.693 of the Keccak-only offload time.

The host path still ends the operation, and the host-process CPU time of 1C+NPU stays within 4 % of its wall-clock time on the SM8650. The check phase, the absorb of H_2, and the opening run on the host alone, and the part of H_2 after the check phase stays on the host path, because H_2 is a serial sponge (Subsection 5.2).

6.4 Energy

We measure energy only on the SM8650, because the measurement switches the CPU governor between an idle phase and the run, which needs root. The energy series run on battery power with USB disconnected, the screen off, and the performance governor on all eight cores. Each run starts only if the battery holds at least 40 % and the CPU and the battery are below 45 °C and 40 °C. A run is one process that repeats one operation for about 6 s on one fixed key and message, and the per-key cache of Subsection 3.1 therefore serves every operation. We read the battery current from the PM8550B charger IC. The energy consumption of an operation is the battery power during the run above the idle power of the tablet under the same governor, times the median operation time.

Each comparison pairs the trials of one series of six, with at least four valid pairs, and every trial runs each configuration once in a cyclic order. Every trial also runs a second, identical 2C process, and the ratio of the two 2C processes sets a noise width of 3 to 7 %. An energy ratio counts as lower or higher only if its 95 % interval excludes 1 and the ratio differs from 1 by more than the noise width, and as within noise or inconclusive otherwise.

On the SM8650, 1C+NPU uses 13 to 27 % less energy per signing operation than 2C at every parameter set (Table 10), with ratios of 0.869, 0.833, and 0.732. In the same trials, 1C+NPU takes 0.744, 0.700, and 0.672 of the 2C signing time. The energy saving therefore grows with the time saving, but it stays smaller than the time saving, because 1C+NPU draws more power than 2C. 2C+NPU uses 1.008, 0.967, and 0.934 of the 3C energy, which is inconclusive for 128s and 192s and lower for 256s. No NPU configuration

Table 10: Energy consumption per operation in mJ on the SM8650 on battery power, median over six trials.

Operation	Configuration	128s	192s	256s
signing	1C	7.34	18.76	45.50
	2C	6.65	16.12	37.87
	3C	6.03	15.09	34.51
	1C+NPU	**5.79**	**13.53**	**27.88**
	2C+NPU	6.04	14.41	32.86
	Ref	29.12	98.64	191.80
verification	1C	7.47	18.96	42.09
	2C	5.60	13.84	31.73
	3C	5.36	13.64	34.82
	1C+NPU	**6.31**	**14.59**	**31.36**
	2C+NPU	6.38	13.99	29.49
	Ref	30.14	101.74	185.97

uses measurably more energy in signing than the CPU configuration with one more core.

In verification, the NPU saves less time, and the energy result depends on the parameter set. For 128s, 1C+NPU uses 1.135 of the 2C energy and 2C+NPU uses 1.177 of the 3C energy, both higher, and two further verification series give the same labels. For 192s both comparisons are inconclusive, and for 256s 2C+NPU uses 0.838 of the 3C energy. In a diagnostic series, a working space in DDR memory instead of the VTCM makes the 1C+NPU operation 1.050 and 1.592 times longer for 128s and 256s, and it uses 1.033 and 1.441 times the energy.

No external meter calibrates the charger IC, and the comparison of the two identical 2C processes is therefore the only check of the reading. The signing comparisons rest on one series of six trials, in which the sustained load heats the CPU to 90 to 99 °C and core 7 runs at 3.053 to 3.245 GHz. A CPU draws more power at a higher clock, and a series at another clock can therefore give other energy ratios.

7 Conclusion

In this paper, we showed that the Hexagon NPU of a mobile SoC shortens one AIMer signing operation, even when the NPU counts as one processor and every configuration runs the same fast host code (Table 8). The dependency graph of the operation shows which work the NPU can compute without the secret key, and the model of the critical path predicts, consistent with the trace, that only work taken from the end of the host path shortens the operation. The linear-map offload therefore moves the map K and the map M of the last repetitions to the NPU, and a trace of both partitions shows that this map M fills NPU time that a Keccak-only offload leaves idle (Figure 4). The host code is a separate contribution, because it makes one CPU core several times faster than the reference implementation (Table 7).

The signing result holds on three SoCs, and the same components accelerate verification, but on the SM8750, 2C+NPU does not verify 128s and 192s faster than 3C, and each SoC needs its own search. On battery power on the SM8650, the shorter signing operation does not use measurably more energy than the CPU configuration with one more core, and in one series 1C+NPU uses less than 2C (Table 10). Verification of 128s uses more energy than the CPU configuration with one more core, because the time ratio does not offset the power ratio. The NPU belongs to the trusted computing base of the signer, and the host still ends the operation with the part of H_2 that no packet overlaps. Future work can shorten host work before H_2 or overlap more of H_2 with NPU work.

References

[A+24] Gorjan Alagic et al. Status report on the first round of the additional digital signature schemes for the NIST post-quantum cryptography standardization process. Technical Report NIST IR 8528, National Institute of Standards and Technology, 2024. doi:10.6028/NIST.IR.8528.

[AB15] Roberto Avanzi and Billy Bob Brumley. Faster 128-EEA3 and 128-EIA3 software. In Yvo Desmedt, editor, *Information Security*, volume 7807 of *Lecture Notes in Computer Science*, pages 199–208. Springer, 2015. doi:10.1007/978-3-319-27659-5_14.

[ABH10] Martin R. Albrecht, Gregory V. Bard, and William Hart. Algorithm 898: Efficient multiplication of dense matrices over GF(2). *ACM Transactions on Mathematical Software*, 37(1):9:1–9:14, 2010. doi:10.1145/1644001.1644010.

[ADKF70] V. L. Arlazarov, E. A. Dinic, M. A. Kronrod, and I. A. Faradzev. On economical construction of the transitive closure of an oriented graph. *Soviet Mathematics Doklady*, 11:1209–1210, 1970.

[Arm26] Arm Ltd. Arm Architecture Reference Manual for A-profile architecture, 2026. ARM DDI 0487, issue M.d. URL: https://developer.arm.com/documentation/ddi0487/latest.

[BBB+24] Slim Bettaieb, Loïc Bidoux, Alessandro Budroni, Marco Palumbi, and Lucas Pandolfo Perin. Enabling PERK and other MPC-in-the-head signatures on resource-constrained devices. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(4):84–109, 2024. doi:10.46586/tches.v2024.i4.84-109.

[BC23] Gustavo Banegas and Florian Caullery. Multi-armed SPHINCS+. In *Applied Cryptography and Network Security Workshops (ACNS 2023, SCI)*, volume 13907 of *Lecture Notes in Computer Science*, pages 500–514. Springer, 2023. Cryptology ePrint Archive, Report 2023/636. doi:10.1007/978-3-031-41181-6_27.

[BDP+12] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Keccak implementation overview, version 3.2. https://keccak.team/files/Keccak-implementation-3.2.pdf, 2012.

[BF26] Ryad Benadjila and Thibauld Feneuil. Breaking the myth of MPCitH inefficiency: Optimizing MQOM for embedded platforms. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2026(3):279–305, 2026. doi:10.46586/tches.v2026.i3.279-305.

[BFU26] Alex Biryukov, Pablo García Fernández, and Aleksei Udovenko. Magic pot: Cryptanalysis of full AIM2 in the standard and related-/reused-key settings using new elimination framework. In *Advances in Cryptology – EUROCRYPT 2026*, volume 16546 of *Lecture Notes in Computer Science*, pages 63–90. Springer, 2026. doi:10.1007/978-3-032-25333-0_3.

[BHK+22] Hanno Becker, Vincent Hwang, Matthias J. Kannwischer, Bo-Yin Yang, and Shang-Yi Yang. Neon NTT: Faster dilithium, kyber, and saber on Cortex-A72 and Apple M1. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(1):221–244, 2022. doi:10.46586/tches.v2022.i1.221-244.

[BK22] Hanno Becker and Matthias J. Kannwischer. Hybrid scalar/vector implementations of Keccak and SPHINCS+ on AArch64. In *Progress in Cryptology – INDOCRYPT 2022*, LNCS, pages 272–293. Springer, 2022. doi:10.1007/978-3-031-22912-1_12.

[BN20] Carsten Baum and Ariel Nof. Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In *PKC*, pages 495–526. Springer, 2020. doi:10.1007/978-3-030-45374-9_17.

[Cau25] Florian Caullery. A PQC engine you didn't know you had: Exploring DSP for cryptographic acceleration. In *2nd International Conference on Cryptography and Information Security (VCRIS)*. IEEE, 2025. doi:10.1109/VCRIS68011.2025.11250569.

[CDG+17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *ACM CCS*, pages 1825–1842. ACM, 2017. doi:10.1145/3133956.3133997.

[CHK+26] Jihoon Cho, Jincheol Ha, Seongkwang Kim, Jihoon Kwon, Byeonghak Lee, Joohee Lee, Jooyoung Lee, Sangyub Lee, Dukjae Moon, Mincheol Son, and Hyojin Yoon. The AIMer signature scheme, version 3.0. Official specification, 2026. URL: https://aimer-signature.org/docs/AIMer-specification-v3%20(Eng).pdf.

[CKM+26] Vu Minh Chau, Nguyen Ngoc Kiet, Pham Quang Minh, Mai Xuan Ngoc, Nguyen Duc Anh, and Hoang Ta. Implementation and optimization of HQC decoding on NPU-integrated devices. arXiv:2606.01968, 2026. URL: https://arxiv.org/abs/2606.01968.

[DHSY24] Sanjay Deshpande, James Howe, Jakub Szefer, and Dongze Yue. SDitH in hardware. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(2):215–251, 2024. doi:10.46586/tches.v2024.i2.215-251.

[FLYL20] Xiongwei Fei, Kenli Li, Wangdong Yang, and Keqin Li. Analysis of energy efficiency of a parallel AES algorithm for CPU–GPU heterogeneous platforms. *Parallel Computing*, 94–95:102621, 2020. doi:10.1016/j.parco.2020.102621.

[Gar25] Gartner. Gartner says worldwide GenAI smartphone end-user spending to total $298 billion by the end of 2025. Gartner press release, 2025. Published September 9, 2025. Accessed October 12, 2026. URL: https://www.gartner.com/en/newsroom/press-releases/2025-09-09-gartner-says-worldwide-generative-artificial-intelligence-smartphone-end-user-spending-to-total-us-dollars-298-billion-by-the-end-of-2025.

[GBL24] Décio Luiz Gazzoni Filho, Guilherme Brandão, and Julio López. Fast polynomial multiplication using matrix multiplication accelerators with applications to NTRU on Apple M1/M3 SoCs. *IACR Communications in Cryptology*, 1(1), 2024. doi:10.62056/a3txommol.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, 1986. doi:10.1145/6490.6503.

[GJJR11] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for side-channel resistance validation. In *NIST Non-Invasive Attack Testing Workshop*, 2011. URL: https://csrc.nist.gov/csrc/media/events/non-invasive-attack-testing-workshop/documents/08_goodwill.pdf.

[HWW+26] Zixu Hao, Jianyu Wei, Tuowei Wang, Minxing Huang, Huiqiang Jiang, Shiqi Jiang, Ting Cao, and Ju Ren. Scaling LLM test-time compute with mobile NPU on smartphones. In *Proceedings of the 21st European Conference on Computer Systems (EuroSys)*, pages 2157–2172. ACM, 2026. doi:10.1145/3767295.3769382.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *STOC*, pages 21–30. ACM, 2007. doi:10.1145/1250790.1250794.

[IT88] Toshiya Itoh and Shigeo Tsujii. A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases. *Information and Computation*, 78(3):171–177, 1988. doi:10.1016/0890-5401(88)90024-7.

[JHH+11] Keon Jang, Sangjin Han, Seungyeop Han, Sue Moon, and KyoungSoo Park. SSLShader: Cheap SSL acceleration with commodity processors. In *8th USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association, 2011. URL: https://www.usenix.org/conference/nsdi11/sslshader-cheap-ssl-acceleration-commodity-processors.

[KEL+26] Ha-Gyeong Kim, Si-Woo Eum, Seung-Won Lee, Min-Ho Song, and Hwa-Jeong Seo. GPU-assisted FAEST v3 signing: Challenge grinding and CPU–GPU pipelining. Cryptology ePrint Archive, Paper 2026/2355, 2026. URL: https://eprint.iacr.org/2026/2355.

[KHS+23] Seongkwang Kim, Jincheol Ha, Mincheol Son, Byeonghak Lee, Dukjae Moon, Joohee Lee, Sangyub Lee, Jihoon Kwon, Jihoon Cho, Hyojin Yoon, and Jooyoung Lee. AIM: Symmetric primitive for shorter signatures with stronger security. In *ACM CCS*, pages 401–415. ACM, 2023. doi:10.1145/3576915.3616579.

[KHSL23] Seongkwang Kim, Jincheol Ha, Mincheol Son, and Byeonghak Lee. Efficacy and mitigation of the cryptanalysis on AIM. Cryptology ePrint Archive, Report 2023/1474, 2023. URL: https://eprint.iacr.org/2023/1474.

[KKW18] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In *ACM CCS*, pages 525–537. ACM, 2018. doi:10.1145/3243734.3243805.

[KLL+25] Jihoon Kwon, Sangyub Lee, Byeonghak Lee, Hwajeong Seo, and Jihoon Cho. Efficient implementations of AIMer post-quantum signature scheme for low-end to high-end IoT devices. *IEEE Internet of Things Journal*, 12(22):46817–46837, 2025. doi:10.1109/JIOT.2025.3597415.

[KMRV18] Angshuman Karmakar, Jose Maria Bermudo Mera, Sujoy Sinha Roy, and Ingrid Verbauwhede. Saber on ARM: CCA-secure module lattice-based key encapsulation on ARM. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):243–266, 2018. doi:10.46586/tches.v2018.i3.243-266.

[Kpq26] KpqC. KpqC algorithms. KpqC research group, 2026. URL: https://www.kpqc.or.kr/contents/03_exhibit/sub_03.html.

[KSX+23] Rabimba Karanjai, Sangwon Shin, Wujie Xiong, Xinxin Fan, Lin Chen, Tianwei Zhang, Taeweon Suh, Weidong Shi, Veronika Kuchta, Francesco Sica, and Lei Xu. TPU as cryptographic accelerator. arXiv:2307.06554, 2023.

[KZ20] Daniel Kales and Greg Zaverucha. Improving the performance of the Picnic signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(4):154–188, 2020. doi:10.13154/tches.v2020.i4.154-188.

[KZ22] Daniel Kales and Greg Zaverucha. Efficient lifting for shorter zero-knowledge proofs and post-quantum signatures. Cryptology ePrint Archive, Report 2022/588, 2022. URL: https://eprint.iacr.org/2022/588.

[Lan10] Adam Langley. ctgrind: Checking that functions are constant time with Valgrind. https://github.com/agl/ctgrind, 2010.

[LE26] Eric Lynch and Dom Elliott. Security for the quantum era: Implementing post-quantum cryptography in Android. Google Blog, 2026. Published March 25, 2026. Accessed October 7, 2026. URL: https://blog.google/security/security-for-the-quantum-era-implementing-post-quantum-cryptography-in-android/.

[LEK+26] Seung-Won Lee, Si-Woo Eum, Min-Seo Kim, Su-Been Cho, and Hwa-Jeong Seo. Hardware-adapted SIMD optimization of AIMer v3 on x86 and Cortex-M55. Cryptology ePrint Archive, Paper 2026/2169, 2026. URL: https://eprint.iacr.org/2026/2169.

[LES26] Seung-Won Lee, Si-Woo Eum, and Hwa-Jeong Seo. Accelerating the AIMer post-quantum signature with AVX-512: A Field–Keccak speedup analysis. Cryptology ePrint Archive, Paper 2026/1417, 2026. URL: https://eprint.iacr.org/2026/1417.

[LJK+23] Minwoo Lee, Kyungbae Jang, Hyeokdong Kwon, Minjoo Sim, Gyeongju Song, and Hwajeong Seo. High-speed implementation of AIM symmetric primitives within AIMer digital signature. Cryptology ePrint Archive, Report 2023/1151, 2023. URL: https://eprint.iacr.org/2023/1151.

[LSKY19] Sokjoon Lee, Hwajeong Seo, Hyeokchan Kwon, and Hyunsoo Yoon. Hybrid approach of parallel implementation on CPU–GPU for high-speed ECDSA verification. *The Journal of Supercomputing*, 75(8):4329–4349, 2019. doi:10.1007/s11227-019-02744-6.

[LSZH22] Wai-Kong Lee, Hwajeong Seo, Zhenfei Zhang, and Seong Oun Hwang. TensorCrypto: High throughput acceleration of lattice-based cryptography using tensor core on GPU. *IEEE Access*, 10:20616–20632, 2022. doi:10.1109/ACCESS.2022.3152217.

[Nei20] Moritz Neikes. TIMECOP: Automated dynamic analysis for timing side-channels. https://www.post-apocalyptic-crypto.org/timecop/, 2020.

[Qua23] Qualcomm Technologies, Inc. Snapdragon 8 Gen 3 mobile platform product brief, 2023. 87-71408-1 Rev. G. URL: https://docs.qualcomm.com/doc/87-71408-1/87-71408-1_REV_G_Snapdragon_8_gen_3_Mobile_Platform_Product_Brief.pdf.

[Qua24a] Qualcomm Technologies, Inc. fastrpc: FastRPC userspace library. GitHub repository README, 2024. URL: https://github.com/qualcomm/fastrpc.

[Qua24b] Qualcomm Technologies, Inc. *Qualcomm Hexagon V75 HVX Programmer's Reference Manual*, 2024. 80-N2040-58 Rev. AB.

[Qua25] Qualcomm Technologies, Inc. Hexagon SDK 6.5.0.0 documentation. Distributed with the Hexagon SDK 6.5.0.0, Community Version, 2025. URL: https://docs.qualcomm.com/nav/home/hexagon-dsp-sdk-setup.html?product=1601111740010422.

[Qua26] Qualcomm Technologies, Inc. Qualcomm Hexagon NPU. Qualcomm product and architecture overview, 2026. URL: https://www.qualcomm.com/processors/hexagon.

[RBV17] Oscar Reparaz, Josep Balasch, and Ingrid Verbauwhede. Dude, is my code constant time? In *DATE*, pages 1697–1702. IEEE, 2017. doi:10.23919/DATE.2017.7927267.

[Sam26] Samsung SDS Open Source. AIMer version 3.0 reference implementation. Official source repository, commit 734cc76fa4695ca8c8eaa070508db8ff62205e68, 2026. URL: https://github.com/samsungsds-opensource/AIMer/tree/734cc76fa4695ca8c8eaa070508db8ff62205e68.

[SKD20] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. Post-quantum authentication in TLS 1.3: A performance study. In *Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2020. doi:10.14722/ndss.2020.24203.

[The26] The mlkem-native project authors. mlkem-native. https://github.com/pq-code-package/mlkem-native, 2026. Commit b87425b.

[WDX+25] Wen Wu, Jiankuo Dong, Zhen Xu, Zhenjiang Dong, Dung Duong, Fu Xiao, and Jingqiang Lin. Symphony of speeds: Harmonizing classic McEliece cryptography with GPU innovation. Cryptology ePrint Archive, Report 2025/748, 2025. URL: https://eprint.iacr.org/2025/748.

[ZZF+24] Tian Zhou, Fangyu Zheng, Guang Fan, Lipeng Wan, Wenxu Tang, Yixuan Song, Yi Bian, and Jingqiang Lin. ConvKyber: Unleashing the power of AI accelerators for faster Kyber with novel iteration-based approaches. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(2):25–63, 2024. doi:10.46586/tches.v2024.i2.25-63.