Accelerating the AIMer Post-Quantum Signature with AVX-512: A Field–Keccak Speedup Analysis

Seung-Won Lee[1 [0009−0009−9583−1387]], Si-Woo Eum[1 [0000−0002−9583−5427]], and Hwa-Jeong Seo[1 [0000−0003−0069−9061]]

Department of Information Computer Engineering,
Hansung University, Seoul, 02876 Korea
dkajdfhd1@gmail.com, shuraatum@gmail.com, hwajeong84@gmail.com

Abstract. AIMer is a post-quantum digital signature scheme with a conservative design. Its security relies only on the symmetric-key one-way function AIM2 and an MPC-in-the-Head (MPCitH) zero-knowledge proof. AIMer is a Korean post-quantum cryptography (KpqC) standard. However, the AIMer standard code released in January 2026 is a portable C reference implementation. It does not include processor-specific optimizations. As a result, it does not exploit AVX-512, a 512-bit vector instruction set available on modern processors. This paper presents an AVX-512 optimized implementation for all six AIMer variants. We parallelize binary-field multiplication with VPCLMULQDQ. The implementation processes four MPC parties per 512-bit register. We also handle conditional XOR accumulation in the linear layer with a single VPTERNLOGQ instruction. In addition, we accelerate the Keccak permutation using AVX-512VL. Our implementation is 1.60–1.84× faster than the AVX2 implementation for signing. We further analyze the source of this speedup. Contrary to common intuition, most of the improvement comes from Keccak. In the 128-bit and 192-bit variants, Keccak accounts for about 86% of the total improvement. Only in the 256-bit variants does the field-arithmetic contribution increase to about 42%, due to the wider field size. These results show that vectorizing symmetric primitives is as important as binary-field arithmetic in MPCitH signatures. In some cases, it is even more important.

Keywords: Post-Quantum Cryptography · AIMer · MPC-in-the-Head · AVX-512 · VPCLMULQDQ · Binary-Field Arithmetic · Keccak · Benchmarking Methodology

1 Introduction

As the threat posed by quantum computers becomes increasingly realistic, the migration from conventional public-key cryptography to post-quantum cryptography (PQC) is rapidly progressing. For this transition to be deployed in real-world systems, not only security but also implementation efficiency is required.

AIMer is a digital signature scheme selected as a Korean post-quantum cryptography (KpqC) standard. Many PQC signature schemes rely on mathematical structures such as lattices or multivariate quadratic equations. In contrast, AIMer derives its security solely from a symmetric-key one-way function and an MPC-in-the-Head (MPCitH) zero-knowledge proof [18]. The primary challenge of such a symmetric-primitive-based design is its computational cost. However, the computation is inherently parallel. In particular, the signer repeatedly performs *the same binary-field arithmetic* for a large number of virtual MPC parties. The entire process is tied together by *a large number of Keccak permutations*. Both components are ideal targets for wide vector instructions (SIMD). This observation motivates the work presented in this paper.

Despite this opportunity, the AIMer implementation released in January 2026 provides only a portable C reference implementation [10]. KpqClean, which collects implementations of KpqC candidate algorithms, includes AVX2-based optimizations. However, it does not yet exploit AVX-512, a 512-bit SIMD instruction set available on modern processors. This instruction set includes VPCLMULQDQ for binary polynomial multiplication and VPTERNLOGQ for three-input bitwise logic. In this work, we integrate both the reference and AVX2 implementations into the liboqs (Open Quantum Safe) framework. Liboqs is an open-source library that provides a common interface for a wide range of post-quantum cryptographic algorithms. We use these implementations as baselines and extend them with AVX-512 optimizations.

To address the issues outlined above, this paper makes the following contributions:

1. We present the first comprehensive AVX-512 implementation covering all six AIMer variants. The implementation includes a VPCLMULQDQ-based four-party field layer, a VPTERNLOGQ-based linear layer, and an AVX-512VL-based Keccak implementation. Compared with the AVX2 implementation, it achieves a speedup of 1.60–1.84×.
2. We show that the observed speedup originates primarily from Keccak rather than field arithmetic, contrary to common intuition. For the 128-bit and 192-bit variants, approximately 86% of the total improvement is attributable to Keccak. Only for the 256-bit variants does the contribution of field arithmetic increase substantially, reaching approximately 42%.
3. We analyze the variation in AVX-512 efficiency across AIMer variants from the perspectives of field width and the number of MPC parties. Elements in $\mathrm{GF}(2^{128})$ and $\mathrm{GF}(2^{256})$ can be packed efficiently into 512-bit lanes, resulting in higher gains. In contrast, $\mathrm{GF}(2^{192})$ incurs losses due to zero padding. Furthermore, the small variants benefit more from four-party packing because the packing operation is repeated more frequently as the number of parties increases.

The remainder of this paper is organized as follows. Section 2 reviews the background of the AIMer signature scheme, the AIM2 one-way function, and the MPCitH proof system. Section 3 surveys prior work on MPCitH signatures and SIMD-based cryptographic optimization. Section 4 describes our AVX-512 implementation, including VPCLMULQDQ-based four-party field arithmetic, a VPTERNLOGQ-based linear layer, and

AVX-512VL-accelerated Keccak. Section 5 quantitatively evaluates performance relative to AVX2 and analyzes the observed improvement by decomposing it into field-arithmetic and Keccak contributions. Finally, Section 6 concludes the paper and discusses limitations and future research directions.

2 Background

AIMer is a post-quantum digital signature scheme. Its security relies on a symmetric-key one-way function. This one-way function is called AIM2 and is defined over the binary field $\mathrm{GF}(2^\lambda)$. A one-way function is easy to evaluate, but it is difficult to recover its input from its output. In this sense, it is similar to a hash function.

The secret key is a randomly chosen input value. The public key is the output obtained by applying AIM2 to this input. A signature proves that the signer knows the input value that generated the public key. At the same time, it does not reveal the secret input itself. This type of proof, which proves only the fact of knowledge without revealing the secret, is called a **zero-knowledge proof**. AIMer generates signatures using a zero-knowledge proof [13].

AIMer constructs this proof using the **MPC-in-the-Head (MPCitH)** technique [24]. It simulates a setting in which N virtual parties hold shares of the secret key and jointly compute AIM2. The verifier then checks this simulated computation. A single execution still leaves room for cheating. Therefore, the same process is repeated several dozen times, denoted by τ in Table 1.

Two types of operations dominate the signing cost. First, the N parties repeatedly perform *the same binary-field arithmetic*. Second, *Keccak hashing* [6,7] is used extensively throughout the process for commitments and challenge generation. This paper accelerates these two components using VPCLMULQDQ and AVX-512VL (Vector Length) [1], respectively. The parameters of each variant are shown in Table 1.

Table 1. Parameters of AIMer variants. λ: security bits/field size, N: number of MPC parties, τ: number of parallel repetitions.

Variant	Security level	Field $\mathrm{GF}(2^\lambda)$	N	τ	Signature (bytes)
AIMER-128f	NIST 1	$\mathrm{GF}(2^{128})$	16	33	5,888
AIMER-128s	NIST 1	$\mathrm{GF}(2^{128})$	256	17	4,160
AIMER-192f	NIST 3	$\mathrm{GF}(2^{192})$	16	49	13,056
AIMER-192s	NIST 3	$\mathrm{GF}(2^{192})$	256	25	9,120
AIMER-256f	NIST 5	$\mathrm{GF}(2^{256})$	16	65	25,120
AIMER-256s	NIST 5	$\mathrm{GF}(2^{256})$	256	33	17,056

3 Related Work

Prior work related to this paper can be divided into two lines. The first line is *MPCitH-based signatures*, to which AIMer belongs. This line shows how signatures have been made smaller and faster by choosing suitable symmetric primitives and checking techniques [19]. The second line is SIMD-based cryptographic optimization, which this paper applies to AIMer. This line focuses on accelerating cryptographic operations using vector instructions. We review these two lines in order.

3.1 MPCitH-Based Signatures

MPC-in-the-Head (MPCitH) is a zero-knowledge proof paradigm proposed by Ishai et al. [14]. In this paradigm, the proof size is proportional to the number of *multiplications*, or nonlinear gates, in the circuit. Subsequent studies have reduced this cost in several directions.

First, Picnic adopted LowMC as a one-way function. LowMC is a block cipher with a small number of AND gates, which reduces the number of multiplications itself [8]. Second, later work introduced techniques that trade communication cost and signature size by using a preprocessing model and adjusting the number of parties N. These techniques reduced signatures to the range of several tens of kilobytes [16]. Third, Banquet used the standard block cipher AES instead of LowMC. Instead of verifying inverse-S-box multiplications one by one, it grouped them into polynomial identities and checked them at once using an amortized check. This reduced the cost of multiplication verification [4]. BN++ further simplified this checking procedure [15]. This last approach was adopted by AIMer.

Furthermore, the FAEST digital signature scheme uses VOLE-in-the-Head [3] instead of MPCitH. It efficiently generates correlated randomness and reduces AES-based signatures to about 5 KB. FAEST advanced to the second round of the NIST additional signature competition [2].

In this line of work, AIMer is a case that introduces a *dedicated* one-way function, AIM. AIM obtains short signatures by minimizing the number of multiplications in the circuit with a low-multiplication S-box. After AIM was published, it was subjected to algebraic analysis [22]. AIM2, a strengthened version reflecting this analysis, was then selected as a KpqC standard [17]. Another line that obtains security only from symmetric primitives is the hash-based signature family SPHINCS+/ SLH-DSA [5,23].

3.2 SIMD-Based Cryptographic Optimization

Cryptographic acceleration using SIMD has been actively studied [9]. Drucker et al. used the fact that 512-bit VPCLMULQDQ performs four independent 64×64 carry-less multiplications in a single instruction. With this property, they substantially improved the throughput of binary-field multiplication and GHASH compared with PCLMULQDQ [11]. The

same authors also proposed VAES. VAES vectorizes AES rounds by packing four blocks into one register. This accelerates parallel modes such as AES-GCM [12].
SIMD optimization has also been studied for lattice-based PQC. For example, AVX-512 vectorization of Dilithium has been reported [25]. For Korean standard cryptographic algorithms, AVX-512 acceleration has been reported for the LSH hash function and LEA-128-GCM. LEA-GCM combines 16-block parallel CTR mode with four-block VPCLMULQDQ-based GHASH. It achieved about a 3.3× improvement over the KISA SSE reference implementation. LSH achieved an average speedup of 2.28× using dual-message processing based on 512-bit register interleaving [21].
The closest prior work to the MPCitH setting is by Lee et al. They accelerated FAEST signatures using ARM NEON AES instructions and parallel VOLE generation [20]. However, most existing studies focus on a single algorithm or a specific platform. To the best of our knowledge, no prior work has comprehensively applied AVX-512 to the MPCitH signature scheme AIMer. This paper fills that gap.

4 Implementation

As discussed in Section 2, the cost of AIMer is concentrated in binary-field arithmetic repeated across all parties and in a large number of Keccak calls. This section reduces these costs using AVX-512. The basic idea consists of two parts. The first is to place the data of multiple parties side by side in a 512-bit register and process them simultaneously with a single instruction. The second is to merge several bit operations into a single instruction.
The existing AVX2 implementation, which we use as the baseline, stores the element of only *one party* in a 128-bit register. It then processes one party at a time using PCLMULQDQ. This is a limitation imposed by the 128-bit register width. In this section, we explain how this limitation is overcome with 512-bit registers. We describe binary-field multiplication in Section 4.1, the linear layer in Section 4.2, and Keccak in Section 4.3.

4.1 Binary-Field Arithmetic: Four-Party Parallel Multiplication per 512-Bit Register

An element of $\mathrm{GF}(2^{128})$ is 128 bits long. Thus, it fits exactly into one 128-bit *lane*. A 512-bit ZMM register can store *four* such lanes. Therefore, elements from four different parties can be loaded side by side into a single register. The key instruction, VPCLMULQDQ, multiplies these four lanes independently without cross-lane interference. As a result, four $64 \times 64 \rightarrow 128$ carry-less multiplications, one for each party, are completed with a single instruction. This contrasts with AVX2, which requires four instructions to perform the same work.
When two 128-bit elements are multiplied, the result can grow up to 256 bits. However, an element of the field $\mathrm{GF}(2^{128})$ must always be 128 bits long. Therefore, the upper part of the product is reduced back to 128

bits according to the irreducible polynomial. This modular reduction is also applied identically to the four lanes, so the four parties are processed together.
Squaring and multiply-accumulate operations are also grouped in units of four parties. Since squaring multiplies a value by itself, the intermediate cross terms vanish. Therefore, it is completed with two carry-less multiplications. In sections where several products must be added consecutively, modular reduction is not performed after every multiplication. Instead, the products are accumulated first, and reduction is performed only once at the end. Modular reduction is an expensive operation. By delaying it and applying it once, its cost can be amortized over multiple products.
Elements of $\mathrm{GF}(2^{192})$ and $\mathrm{GF}(2^{256})$ occupy 1.5 and 2 128-bit lanes, respectively. The 256-bit case uses exactly two lanes. Thus, four parties fit into two ZMM registers without unused space. In contrast, the 192-bit case requires 1.5 lanes. The remaining 0.5 lane is filled with zero padding. Therefore, part of the lane capacity is wasted in the 192-bit variants.

4.2 Linear Layer: Combining Two Operations into One Instruction with VPTERNLOGQ

The linear layer multiplies an input vector by a binary matrix. The result is equal to the XOR of all rows whose corresponding input bits are 1. Therefore, the implementation scans the input bits one by one. If a bit is 1, the corresponding row is XORed into the accumulator. If a bit is 0, the row is skipped.
The issue is the conditional branch implied by "if the bit is 1." Branching is inefficient in SIMD code. Therefore, we replace the branch with *masking*. The branch-free procedure is as follows. First, one input bit is expanded into a value filled with the same bit. If the bit is 1, the expanded value consists entirely of 1s. If the bit is 0, it consists entirely of 0s. When this mask is ANDed with the row, the row remains unchanged if the bit is 1. If the bit is 0, the result becomes all zero. This value is then XORed into the accumulator.
The key point is that the computation "AND the mask with the row and then XOR the result into the accumulator" is a single *three-input bitwise logic* operation. In AVX2, this requires *two instructions*, AND and XOR. In contrast, **VPTERNLOGQ** in AVX-512 evaluates an arbitrary three-input bitwise logic function in *one instruction*.

4.3 Keccak: AVX-512VL Acceleration

We accelerate the Keccak permutation using the same two ideas described above. In this case, we use the **AVX-512VL** extension. AVX-512VL allows AVX-512 instructions to be used with 256-bit *YMM* registers.

Algorithm 1 Linear-layer accumulation: mask generation and VPTERNLOGQ

Require: Party element $a = (h \,\|\, l)$ (upper and lower 64 bits), matrix rows $\mathrm{row}_0, \ldots, \mathrm{row}_{127}$, accumulator k
(h, l, k, row_j *are 512-bit registers whose 64-bit lanes contain four parties*)

1: **for** $j \leftarrow 63$ **downto** 0 **do** ▷ Upper 64 bits
2: mask $\leftarrow$ Asr(h, 63) ▷ Arithmetic right shift by 63: replicate the most significant bit to the full 64-bit word ($1 \rightarrow$ `1⋯1`, $0 \rightarrow$ `0⋯0`)
3: $k \leftarrow$ Vpternlogq(k, mask, row_{64+j}, `0x78`) ▷ $k \oplus (\mathrm{mask} \wedge \mathrm{row})$, AND+XOR in one instruction
4: $h \leftarrow$ Shl(h, 1) ▷ Move the next bit to the most significant position
5: **end for**
6: **for** $j \leftarrow 63$ **downto** 0 **do** ▷ Lower 64 bits, repeated identically with l and row_j
7: mask $\leftarrow$ Asr(l, 63); $k \leftarrow$ Vpternlogq(k, mask, row_j, `0x78`); $l \leftarrow$ Shl(l, 1)
8: **end for**
9: **return** k

Packing Multiple Hashes into One Register AIMer invokes the same Keccak permutation many times. Therefore, as shown in Fig. 1, we interleave four Keccak states. The same 64-bit lane from each state is placed side by side in one 256-bit YMM register, giving $4 \times 64 = 256$ bits per lane group. Then, a single instruction processes the corresponding lanes of four states at the same time. As a result, four hash computations proceed in parallel.

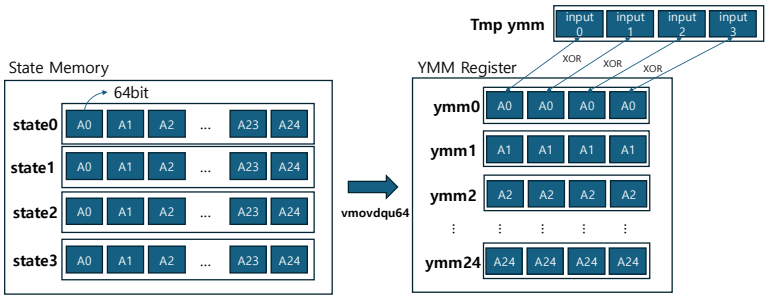

Fig. 1. The same lanes from four Keccak states are collected into one 256-bit YMM register. The blue A0 lanes from the four states form ymm0.

Merging Multiple Operations into One Instruction Each round of the Keccak permutation consists of five steps. Among them, the dominant operations are *bit rotations* and *logical operations*. The rotation

step rotates each 64-bit lane by a fixed amount. AVX2 implements this with three instructions: left shift, right shift, and OR. In contrast, `vprolq` performs the same operation with a single instruction.
The three XOR operations in the diffusion step and the nonlinear operation $a \oplus (\neg b \wedge c)$ are both *bitwise logic operations over three inputs*. Therefore, each of them can be handled by one VPTERNLOGQ instruction.

5 Performance

This section quantitatively evaluates the proposed AVX-512 implementation. We first describe the measurement environment. We then present the speedup over AVX2 for all six AIMer variants. Finally, we decompose the speedup and analyze whether it comes from field arithmetic or Keccak.

5.1 Measurement Environment

The measurements were conducted on an 11th-generation Intel Core i7-1165G7 processor with Ubuntu 24.04 and gcc 13.3.0. The code was compiled with `-O3 -fomit-frame-pointer`. The CPU governor was fixed to `performance`. For reproducibility, **Turbo Boost was disabled**. The benchmark was pinned to a single core using `taskset`. We report the median over 50 measurements for each operation.
The measurement environment is summarized in Table 2.

Table 2. Measurement environment

CPU	11th-generation Intel Core i7-1165G7 (Tiger Lake, base 2.8 GHz)
ISA	AVX-512/VPCLMULQDQ support
OS	Ubuntu 24.04
Compiler	gcc 13.3.0
CFLAGS	`-O3 -fomit-frame-pointer`
CPU governor	Fixed to `performance`
Turbo Boost	**Disabled** for reproducibility
Core pinning	Pinned to a single core using `taskset`
Reporting	Median over 50 measurements per operation

5.2 Performance Evaluation: AVX2 vs. AVX-512

Table 3 shows the median cycle counts and speed ratios of the AVX2 and AVX-512 implementations for signing and verification. For signing, AVX-512 is 1.60–1.84× faster than AVX2. For verification, it is 1.58–1.86× faster.
The speed ratio differs across variants. This can be explained by two factors. First, at the same security level, the small variants (N=256)

Table 3. AVX2 vs. AVX-512 performance. Values are median cycles over 50 measurements with Turbo Boost disabled. Ratio denotes the AVX2/AVX-512 speed ratio. (*=ours)

	Signing			Verification		
Variant	AVX2	***AVX-512**	Ratio	AVX2	***AVX-512**	Ratio
128f	1,346,647	774,954	1.74	1,322,628	784,276	1.69
128s	10,024,946	5,434,747	1.84	9,888,369	5,303,667	1.86
192f	3,391,874	2,117,497	1.60	3,392,433	2,151,221	1.58
192s	25,234,828	15,185,253	1.66	25,051,151	15,023,986	1.67
256f	6,685,139	3,872,719	1.73	6,665,384	3,881,205	1.72
256s	47,950,861	26,097,011	1.84	47,656,786	25,852,180	1.84

show larger improvements than the fast variants (N=16). As the number of parties increases, four-party packing is repeated more often, and its benefit accumulates. Second, the 192-bit variants show the smallest improvements. As discussed in Section 4.1, an element of $\mathrm{GF}(2^{192})$ occupies 1.5 lanes, so part of the lane capacity is wasted due to zero padding. In contrast, 128-bit and 256-bit elements fit into lanes without unused space.

A more important question is whether this improvement comes from field arithmetic or Keccak. To separate the two effects, we fixed the Keccak backend to AVX-512VL and changed only the field implementation between AVX2 and AVX-512. The resulting speed ratios are shown in Table 4. The contribution of each component to the overall improvement is shown in Fig. 2.

The field-only speedup is only 1.08–1.11× for the 128-bit and 192-bit variants. This shows that the effect of field optimization alone is small. It also shows that about 86% of the total improvement comes from Keccak. Once the field arithmetic has already been vectorized with AVX2, Keccak becomes the dominant cost. Therefore, the largest benefit of AVX-512 comes from the Keccak permutation. The exception is the 256-bit case. In this case, the field width is larger, and field arithmetic accounts for a larger fraction of the cost. In addition, each element is packed exactly into two lanes. As a result, the field-only speedup increases to 1.30–1.36×, and the field contribution grows to about 42%.

6 Conclusion

This paper presented an AVX-512 optimized implementation for all six variants of AIMer, a KpqC standard signature scheme. We parallelized binary-field multiplication across four parties per 512-bit register using VPCLMULQDQ. We also handled conditional XOR accumulation in the linear layer using VPTERNLOGQ. In addition, we accelerated Keccak using AVX-512VL. As a result, our implementation achieved a 1.60–1.84× speedup over AVX2 for signing.

Table 4. Speed ratios by variant. "Overall" denotes the AVX-512 speedup over AVX2. "Field only" denotes the speed ratio when the Keccak backend is fixed to AVX-512VL and only the field implementation is changed. The difference between the two corresponds to the Keccak contribution.

Variant	Overall	Field only
128f	1.74	1.11
128s	1.84	1.11
192f	1.60	1.08
192s	1.66	1.09
256f	1.73	1.30
256s	1.84	1.36

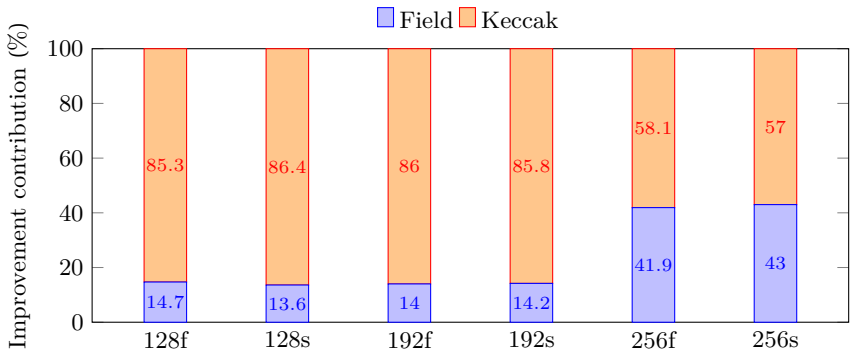

Fig. 2. Contribution of field arithmetic and Keccak to the overall speedup. Keccak accounts for about 86% of the improvement in the 128-bit and 192-bit variants, whereas the field contribution increases to about 42% only in the 256-bit variants.

We further decomposed this improvement into the contributions of field arithmetic and Keccak. The results show that, in the 128-bit and 192-bit variants, about 86% of the improvement comes from Keccak. Only in the 256-bit variants does the field contribution increase to about 42%. This result indicates that vectorizing Keccak is as important as, or even more important than, optimizing field arithmetic in MPCitH signatures. Once the field arithmetic has been vectorized, Keccak becomes the dominant cost.

The measurements in this study were conducted on a single CPU, Tiger Lake. Turbo Boost was disabled, and the benchmarks were pinned to a single core. Therefore, we did not evaluate behavior on other AVX-512 microarchitectures. We also did not examine the effects of frequency throttling caused by heavy AVX-512 usage or the impact of multicore environments.

Based on these limitations, future work includes cross-platform measurements across multiple microarchitectures and an analysis of frequency behavior. It also includes further optimization using AVX-512 IFMA

and related extensions. Another direction is to extend the decomposition methodology used in this paper to other SIMD architectures, such as ARM SVE and NEON. In particular, our analysis showing that Keccak is the bottleneck suggests that future optimization should prioritize symmetric primitives.

References

1. Anderson, C.S., Zhang, J., Cornea, M.: Enhanced vector math support on the intel® avx-512 architecture. In: 2018 IEEE 25th Symposium on Computer Arithmetic (ARITH). pp. 120–124. IEEE (2018)
2. Baum, C., Beullens, W., Braun, L., de Saint Guilhem, C.D., Klooß, M., Majenz, C., Mukherjee, S., Orsini, E., Ramacher, S., Rechberger, C., et al.: Faest v2: Algorithm specifications (2025)
3. Baum, C., Braun, L., de Saint Guilhem, C.D., Klooß, M., Orsini, E., Roy, L., Scholl, P.: Publicly verifiable zero-knowledge and post-quantum signatures from vole-in-the-head. In: Annual international cryptology conference. pp. 581–615. Springer (2023)
4. Baum, C., de Saint Guilhem, C.D., Kales, D., Orsini, E., Scholl, P., Zaverucha, G.: Banquet: short and fast signatures from aes. In: IACR International Conference on Public-Key Cryptography. pp. 266–297. Springer (2021)
5. Bernstein, D.J., Hülsing, A., Kölbl, S., Niederhagen, R., Rijneveld, J., Schwabe, P.: The sphincs+ signature framework. In: Proceedings of the 2019 ACM SIGSAC conference on computer and communications security. pp. 2129–2146 (2019)
6. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak sponge function family main document. Submission to NIST (Round 2) **3**(30), 320–337 (2009)
7. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 313–314. Springer (2013)
8. Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Zaverucha, G.: Post-quantum zero-knowledge and signatures from symmetric-key primitives. In: Proceedings of the 2017 acm sigsac conference on computer and communications security. pp. 1825–1842 (2017)
9. Chhugani, J., Nguyen, A.D., Lee, V.W., Macy, W., Hagog, M., Chen, Y.K., Baransi, A., Kumar, S., Dubey, P.: Efficient implementation of sorting on multi-core simd cpu architecture. Proceedings of the VLDB Endowment **1**(2), 1313–1324 (2008)
10. Cho, J., Ha, J., Kim, S., Kwon, J., Lee, B., Lee, J., Lee, J., Lee, S., Moon, D., Son, M., et al.: The aimer signature scheme
11. Drucker, N., Gueron, S., Krasnov, V.: Fast multiplication of binary polynomials with the forthcoming vectorized vpclmulqdq instruction. In: 2018 IEEE 25th Symposium on Computer Arithmetic (ARITH). pp. 115–119. IEEE (2018)

12. Drucker, N., Gueron, S., Krasnov, V.: Making aes great again: the forthcoming vectorized aes instruction. In: 16th International Conference on Information Technology-New Generations (ITNG 2019). pp. 37–41. Springer (2019)
13. Fiege, U., Fiat, A., Shamir, A.: Zero knowledge proofs of identity. In: Proceedings of the nineteenth annual ACM symposium on Theory of computing. pp. 210–217 (1987)
14. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing. pp. 21–30 (2007)
15. Kales, D., Zaverucha, G.: Efficient lifting for shorter zero-knowledge proofs and post-quantum signatures. Cryptology ePrint Archive (2022)
16. Katz, J., Kolesnikov, V., Wang, X.: Improved non-interactive zero knowledge with applications to post-quantum signatures. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 525–537 (2018)
17. Kim, S., Ha, J., Son, M., Lee, B.: Efficacy and mitigation of the cryptanalysis on aim. Cryptol. ePrint Arch., IACR, Bellevue, WA, USA, Rep **1474**, 2023 (2023)
18. Kim, S., Ha, J., Son, M., Lee, B., Moon, D., Lee, J., Lee, S., Kwon, J., Cho, J., Yoon, H., et al.: The aimer signature scheme. Tech. rep., Technical report. NIST (2023)
19. Lee, M., Jang, K., Kwon, H., Sim, M., Song, G., Seo, H.: High-speed implementation of aim symmetric primitives within aimer digital signature. Cryptology ePrint Archive (2023)
20. Lee, S.W., Kim, H.G., Song, M.H., Eum, S.W., Seo, H.J.: Accelerating faest signatures on arm: Neon simd aes and parallel vole optimization. Applied Sciences **16**(8), 3782 (2026)
21. Lee, S.W., Song, M.H., Kim, H.G., Kim, U.J., Eum, S.W., Seo, H.J.: Advanced vector extensions 512 acceleration of lsh and leagcm. Cryptology ePrint Archive (2026)
22. Liu, F., Mahzoun, M., Øygarden, M., Meier, W.: Algebraic attacks on rain and aim using equivalent representations. IACR Transactions on Symmetric Cryptology **2023**(4), 166–186 (2023)
23. Saarinen, M.J.O.: Accelerating slh-dsa by two orders of magnitude with a single hash unit. In: Annual International Cryptology Conference. pp. 276–304. Springer (2024)
24. Delpech de Saint Guilhem, C., Orsini, E., Tanguy, T.: Limbo: efficient zero-knowledge mpcith-based arguments. In: Proceedings of the 2021 acm sigsac conference on computer and communications security. pp. 3022–3036 (2021)
25. Xu, R., He, D., Luo, M., Peng, C., Zeng, X.: Optimizing dilithium implementation with avx2/-512. ACM Transactions on Embedded Computing Systems **23**(6), 1–30 (2024)