Advanced Vector Extensions 512 Acceleration of LSH and LEA-GCM

Seung-Won Lee[1][0009−0009−9583−1387], Min-Ho Song[1][0009−0007−6277−0069], Ha-Gyeong Kim[1][0009−0008−5840−5517], Ui-Jae Kim[1][0009−0004−2617−9659], 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, smino0906@gmail.com, kimhaha4420@gmail.com, chrisvt424@gmail.com, shuraatum@gmail.com, hwajeong84@gmail.com

Abstract. This paper presents high-performance Advanced Vector Extensions 512(AVX-512) implementations of two Korean standard cryptographic algorithms: the Lightweight Secure Hash(LSH) hash function and Lightweight Encryption Algorithm-Galois/Counter Mode(LEA-GCM) authenticated encryption. For LSH, we apply three optimization strategies: single-message processing using AVX-512 512-bit vector registers, dual-message parallel processing through register interleaving, and multi-core parallelization using a dynamic queue-based pthread thread pool. For LEA-GCM, we propose an end-to-end optimization that replaces scalar Counter mode(CTR) encryption with 16-block AVX-512 parallel processing and Streaming SIMD Extensions(SSE)-based Galois Hash(GHASH) authentication with VPCLMULQDQ-based 4-block parallel processing. Performance evaluation on an Intel Core i7-1165G7 (Tiger Lake) processor shows that LSH-256 achieves an average 1.16× throughput improvement and LSH-512 achieves an average 1.61× improvement over the Korea Internet and Security Agency(KISA) AVX2 reference implementation. Dual-message interleaving achieves an average superlinear speedup of 2.28× driven by instruction-level parallelism (ILP), and the 8-core thread pool delivers speedups of 3.50× to 5.12×. The optimized LEA-GCM implementation achieves a 3.26× throughput improvement over the KISA SSE-based reference and a 12.1× improvement over the pure software implementation for 4096-byte inputs, with correctness verified against KISA official test vectors.

Keywords: Advanced Vector Extensions 512, Lightweight Secure Hash, Lightweight Encryption Algorithm, Galois/Counter Mode, Single Instruction Multiple Data, Korean Cryptography, Performance Optimization

1 Introduction

Cryptographic hash functions support the security of modern digital systems by providing data integrity verification, digital signatures, and mes-

sage authentication across network protocols, cloud services, and embedded devices [12]. As the volume of data requiring cryptographic processing grows, performance limits in hash function implementations impose measurable costs on system throughput and latency. The Korea Internet and Security Agency (KISA) LSH algorithm was developed in 2014 and is the Korean national standard hash function. Intel's recent processors support the Advanced Vector Extensions 512 (AVX-512) instruction set, which provides 512-bit Single Instruction Multiple Data (SIMD) registers —twice the width of the previous Advanced Vector Extensions 2 (AVX2) standard. Prior work has demonstrated the effectiveness of AVX-512 on Korean block ciphers such as LEA, HIGHT, and CHAM [5].

Authenticated Encryption, which simultaneously provides confidentiality and integrity, is a core component of modern secure communication protocols. LEA-GCM [8], which combines the Korean standard block cipher LEA with Galois/Counter Mode (GCM), is an authenticated encryption scheme approved by KISA's Korea Cryptographic Module Validation Program (KCMVP) [2] and is used in domestic security products. However, the LEA-GCM reference implementation published by KISA [13] processes Counter mode(CTR) encryption one block at a time using scalar operations and handles Galois Hash(GHASH) authentication with Streaming SIMD Extensions(SSE) PCLMULQDQ (128-bit) instructions, and therefore does not take advantage of AVX-512-level performance. The VPCLMULQDQ (512-bit carry-less multiplication) instruction provided by AVX-512 can process GCM GHASH operations in groups of four blocks, significantly improving GHASH throughput compared to existing SSE-based implementations.

The goal of this study is to design, implement, and evaluate AVX-512-based optimizations for the Korean standard LSH hash function and LEA-GCM authenticated encryption. To this end, we propose three LSH implementation strategies—single-message SIMD processing, dual-message register interleaving, and dynamic thread pool parallelization —alongside an optimized LEA-GCM pipeline combining 16-block parallel CTR mode with 4-block parallel VPCLMULQDQ GHASH. Specifically, this comprehensive evaluation and structural analysis are guided by four central research questions: First, how effectively does the AVX-512 intrinsics-based implementation improve the single-message throughput and cycles per byte (CPB) across all six LSH variants compared to the AVX2 reference? Second, can dual-message simultaneous processing using an interleaved register layout for LSH-256 exploit instruction-level parallelism (ILP) to achieve superlinear speedups that exceed the theoretical linear bound? Third, to what extent does the dynamic queue-based thread pool parallelization scale total throughput and multi-core efficiency across various LSH variants and input sizes? Fourth, how much does combining AVX-512 CTR 16-block parallel encryption with VPCLMULQDQ GHASH 4-block parallel processing quantitatively improve the authenticated encryption performance of LEA-GCM over the KISA references?

By systematically addressing and answering these research questions, this study delivers the following primary contributions:

1. The first systematic AVX-512 implementation of LSH covering all six hash variants, with correctness verified against the KISA reference implementation.
2. Demonstration that a dual-message parallel kernel based on 512-bit register interleaving achieves an average superlinear speedup of 2.28×, exceeding the theoretical 2× linear bound through instruction-level parallelism (ILP).
3. A dynamic queue thread pool implementation that achieves near-linear multi-core scalability for large input sizes, along with a quantitative analysis of scaling limits caused by AVX-512 frequency throttling and memory bandwidth contention.
4. An optimized LEA-GCM implementation combining AVX-512 CTR (16-block parallel) and VPCLMULQDQ GHASH (4-block parallel), achieving 3.3× authenticated encryption throughput improvement over the KISA SSE PCLMULQDQ reference and 12.1× over the software GHASH reference for 4096-byte inputs, with correctness verified against KISA reference outputs.

The remainder of this paper is organized as follows. Section 2 provides background on the LSH algorithm, LEA block cipher, GCM authenticated encryption, AVX-512 instruction set, and related work that establishes the context for the research questions. Section 3 describes the proposed implementation methods, covering the design decisions for four strategies corresponding to single-message, dual-message, thread pool, and LEA-GCM AVX-512 optimizations. Section 4 presents the experimental results and statistical analysis that quantitatively answer the research questions. Section 5 concludes with a summary of findings, a discussion of limitations encountered during the study, and directions for future work.

2 Background

This section provides background knowledge to help readers understand the context of this study. We first introduce the Korean standard cryptographic algorithms that are the targets of optimization: the LSH hash function, the LEA block cipher, and LEA-GCM authenticated encryption. We then summarize the key features of the AVX-512 SIMD instruction set used for optimization, and conclude with a review of related prior work.

2.1 LSH Algorithm

The LSH algorithm is a hash function family standardized by KISA in 2014 and designed for fast software implementation on general-purpose processors. The family is divided into two subfamilies [15]: LSH-256, which operates on 32-bit words and uses a 256-bit internal state, and LSH-512, which operates on 64-bit words and uses a 512-bit internal state. Output lengths of 224, 256, 384, and 512 bits are supported in each subfamily, giving a total of six variants.

The internal state of LSH is shown in Fig.1. It consists of 16 words divided into eight even-chain words ($\mathbf{cv}_e$) and eight odd-chain words ($\mathbf{cv}_o$). The compression function repeats two steps. In the *Mix* step, Addition-XOR-Rotation (AXR) operations are applied to corresponding pairs of words in the even and odd chains. In the *Word Permutation* step, the positions of the 16 words are rearranged according to a predefined permutation. The step constants used at each step are precomputed values specific to each variant. LSH-512 performs 26 rounds and LSH-256 performs 28 rounds.

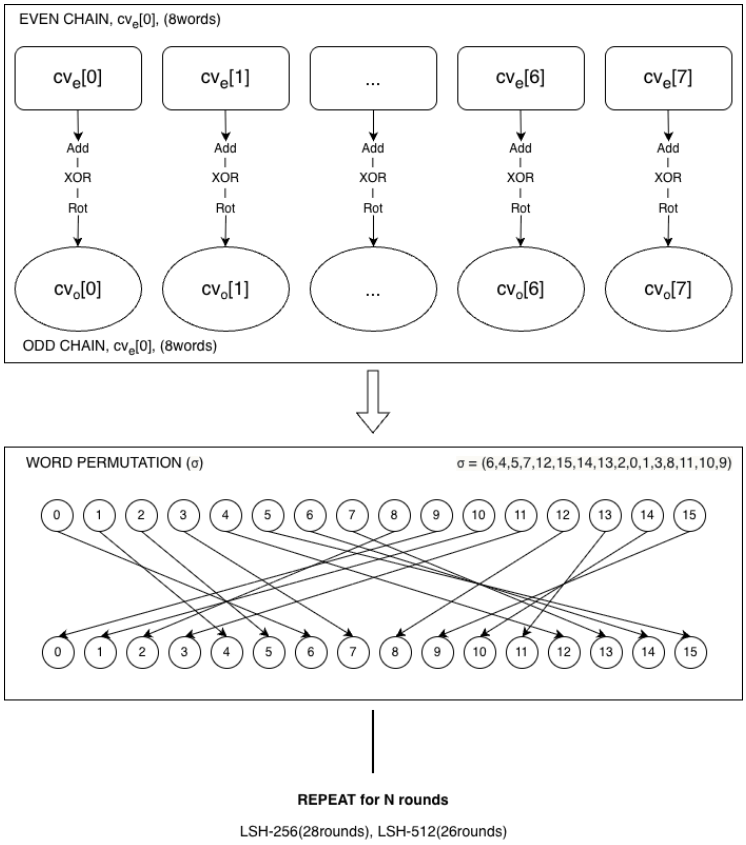

Fig. 1. LSH Compression Function Structure

The suitability of LSH for AVX-512 comes from the size of its internal state. The single chain of LSH-512 (8 words $\times$ 64 bits = 512 bits) maps exactly to one ZMM register, so all operations in the Mix step can be completed within a single register. In contrast, the LSH-256 reference implementation is already efficiently optimized with AVX2, leaving less room for additional gains from moving to AVX-512.

2.2 LEA

LEA is a Korean national standard block cipher developed by KISA in 2013 with the goal of fast encryption in big data, cloud, and mobile environments [8]. LEA adopts an ARX structure consisting only of Addition, Rotation, and XOR without an S-box, enabling efficient implementation on general-purpose processors without hardware lookup tables.

Round Function: LEA splits a 128-bit plaintext into four 32-bit words (X_0, X_1, X_2, X_3) and applies the following XAR (XOR+Add+Rotate) operations repeatedly over 24 rounds:

$$\begin{aligned} X_2' &= \mathrm{ROR}\big((X_2 \oplus RK_4) \boxplus (X_3 \oplus RK_5),\ 3\big) \\ X_1' &= \mathrm{ROR}\big((X_1 \oplus RK_2) \boxplus (X_2 \oplus RK_3),\ 5\big) \\ X_0' &= \mathrm{ROL}\big((X_0 \oplus RK_0) \boxplus (X_1 \oplus RK_1),\ 9\big) \\ X_3' &= X_0 \end{aligned}$$

where $\boxplus$ denotes modular 32-bit addition, ROR/ROL denotes right/left rotation, and RK_0–RK_5 are six round keys. Since all operations are completed at the word level without data dependencies between words, the structure is well suited for SIMD parallelization.
The key schedule generates six 32-bit round keys per round from a 128-bit master key, using a total of $24 \times 6 = 144$ round keys. The key schedule uses predefined delta constants combined with addition and rotation operations.
For AVX-512 parallelization, since the encryption of each block is independent, data parallelism can be exploited. An AVX-512 ZMM register (512 bits) holds sixteen 32-bit elements, so packing the same-position words from 16 blocks into a single ZMM register allows a single XAR operation to process 16 blocks simultaneously [5]. In this study, this 16-block parallel implementation is used as the CTR encryption core for LEA-GCM.

2.3 GCM Authenticated Encryption

Galois/Counter Mode (GCM) is an Authenticated Encryption with Associated Data (AEAD) mode of operation specified in NIST SP 800-38D. GCM provides confidentiality and integrity in a single pass and is widely used in modern security protocols such as TLS 1.3 [16], IPsec, and SSH [6]. LEA-GCM uses LEA as its block cipher and is an authenticated encryption scheme approved by KISA KCMVP [2].
GCM consists of two internal operations [14]. In CTR encryption, a counter block composed of a 96-bit nonce and a 32-bit counter is encrypted by LEA to produce a keystream, which is then XORed with the plaintext to generate the ciphertext. In GHASH authentication, polynomial multiplication over GF(2^{128}) is used to accumulate the additional authentication data (AAD), ciphertext, and length information to produce an authentication tag. Details are shown in the figure below.

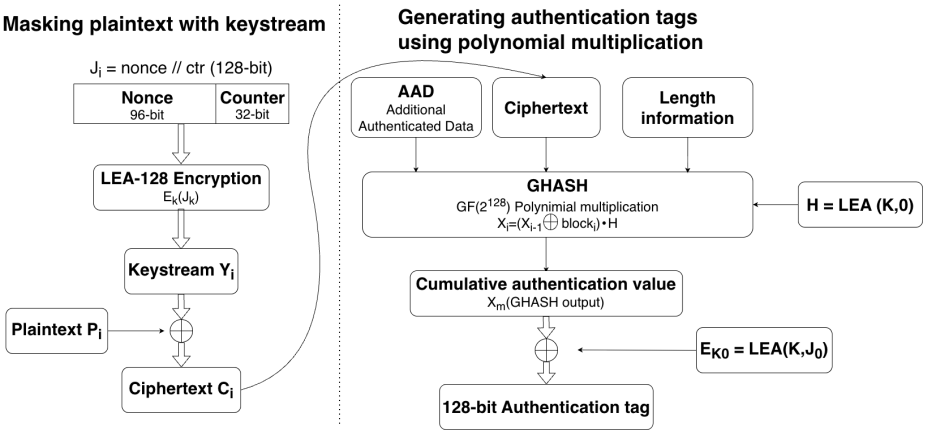

Fig. 2. LEA-GCM Internal Operations. (Left) CTR Encryption: counter block encryption followed by plaintext XOR. (Right) GHASH Authentication: accumulation of AAD and ciphertext via GF(2^{128}) multiplication, then tag generation.

The GF(2^{128}) multiplication in GHASH is expressed as carry-less multiplication, which can be accelerated on x86 architectures using the PCLMULQDQ instruction from SSE4.1 [7]. However, PCLMULQDQ processes only one pair of 64-bit operands in a 128-bit XMM register. The VPCLMULQDQ instruction in AVX-512 uses 512-bit ZMM registers to process four pairs of 64-bit carry-less multiplications in a single instruction, improving GHASH throughput. The KISA LEA-GCM reference implements CTR encryption one block at a time (scalar) and GHASH with PCLMULQDQ (128-bit), and does not provide AVX-512-level acceleration.

2.4 Advanced Vector Extensions 512 Instruction Set

AVX-512 extends Intel's SIMD architecture to 512-bit vector registers (ZMM0–ZMM31), providing 32 registers that are twice the width of AVX2's 256-bit YMM registers [10]. A single AVX-512 instruction can process sixteen 32-bit integers or eight 64-bit integers simultaneously. It also introduces mask registers that enable element-wise conditional operations and new gather/scatter instructions for non-contiguous memory access. AVX-512 is available on Intel Skylake-X, Ice Lake, Tiger Lake, Rocket Lake, and other microarchitectures [9]. A notable characteristic is processor frequency down-clocking (AVX-512 license transition) that occurs when 512-bit operations are executed, which can offset throughput gains especially when multiple cores execute AVX-512 instructions simultaneously.

2.5 Related Work

This section reviews prior work on optimizing cryptographic algorithms using the AVX-512 instruction set, organized into three categories. We

first examine research most directly related to this study—optimization of Korean standard block ciphers—followed by work on ARX-based stream ciphers and authenticated encryption, and finally examples of AVX-512 use in post-quantum cryptography.

AVX-512 Optimization of Korean Standard Block Ciphers: Seo et al. [17] proposed SIMD-based parallel implementations of LEA on ARM processors and demonstrated performance improvements on various embedded platforms. This work provides early analysis of the data parallelism in LEA. Choi et al. [5] proposed AVX-512 parallel implementations of Korean block ciphers LEA, HIGHT, and CHAM, demonstrating up to 1,325.81% performance improvement over the KISA reference implementations. Their study adopts a data-parallel approach that packs same-position words from multiple blocks into AVX-512 512-bit registers, processing 16 blocks for LEA, 64 blocks for HIGHT, and 32 blocks for CHAM-64 simultaneously. However, this work is limited to pure block cipher mode and does not address performance in authenticated encryption modes such as GCM. Our study builds on their LEA AVX-512 parallel encryption routine as the CTR encryption core to optimize the end-to-end performance of LEA-GCM. Choi et al. [4] proposed parallel implementations of the LSH hash function on both CPU and GPU. This work combined thread-level parallelism on multi-core CPUs with large-scale parallel processing on GPUs to improve LSH throughput, and experimentally demonstrated that LSH has a structure well suited for data parallelism.

AVX-512 Optimization of Stream Ciphers and Authenticated Encryption: Käsper and Schwabe [11] proposed a timing-attack-resistant software AES-GCM implementation, laying the groundwork for GHASH optimization. This work presents a method for implementing GCM without secret-dependent memory accesses and serves as a predecessor to subsequent PCLMULQDQ-based GHASH acceleration research. Wang et al. [19] implemented AVX-512 acceleration of the ChaCha20 stream cipher in a CPU-GPU hybrid computing environment, achieving a throughput of 10.05 Gb/s. ChaCha20, like LEA, uses an ARX structure consisting only of addition, XOR, and rotation, and the simultaneous generation of multiple keystream blocks using AVX-512 is the key to its performance improvement.
Bellizia et al. [1] implemented the sponge-based authenticated encryption scheme Spook with AVX-512, achieving 271 cycles per 128-byte block. Like GCM, Spook is an AEAD scheme that provides both confidentiality and integrity in a single pass, and this work shows that SIMD optimization can be applied to various authenticated encryption structures.

AVX-512 Optimization of Post-Quantum Cryptography: Takahashi [18] proposed a parallel Number Theoretic Transform (NTT) implementation using Intel AVX-512 instructions, accelerating a key operation in lattice-based post-quantum cryptography. This work demon-

strates that AVX-512's 512-bit registers are effective for parallel processing of polynomial coefficients. Cheng et al. [3] applied AVX-512 to finite field arithmetic operations in the NIST post-quantum cryptography candidate SIKE, achieving up to 4.61× performance improvement. This work shows that AVX-512 can be broadly applied not only to symmetric-key cryptography but also to public-key operations.
This study presents two contributions that complement the prior work described above. We provide the first systematic AVX-512 implementation of the LSH hash function, and present an end-to-end optimization that combines CTR parallelization with VPCLMULQDQ GHASH acceleration for the LEA-GCM authenticated encryption mode—a combination not addressed in prior work.

3 Proposed Implementation Methods

This section describes four implementation strategies, each corresponding to one of the research questions. Sections 3.1–3.3 target the LSH hash function. Section 3.1 covers single-message throughput improvement using wide AVX-512 registers, Section 3.2 describes simultaneous processing of two messages through register interleaving, and Section 3.3 explains multi-core parallelization using a pthread thread pool. The final section, 3.4, targets LEA-128-GCM authenticated encryption and presents an end-to-end optimization that combines AVX-512 16-block parallel CTR encryption with VPCLMULQDQ 4-block parallel GHASH.

3.1 Single-Message Implementation Using Advanced Vector Extensions 512

The single-message AVX-512 implementation replaces the AVX2-based scalar and 256-bit vector operations of the KISA reference implementation with 512-bit operations, making use of the wider register file. For LSH-512, the eight 64-bit internal state words fit exactly into a single ZMM register, so the Mix and Word Permutation steps can operate on the full state with a single instruction. For LSH-256, the two 128-bit lanes of 32-bit words are packed into a YMM register, and the AVX-512 extension provides additional instruction variants that reduce the number of operations compared to AVX2. Correctness is established by comparing the output of the proposed AVX-512 implementation against the KISA reference implementation for all six variants and multiple input lengths, including an empty message, short strings, and multi-block inputs up to 4096 bytes.

3.2 Dual-Message Parallel Implementation

The dual-message parallel implementation processes two independent messages simultaneously by interleaving their data within AVX-512 registers. Specifically, a 512-bit ZMM register is split into two 256-bit halves, each holding the state of one message. All arithmetic and logical operations of the LSH-256 compression function are applied to both halves

with a single instruction, effectively cutting the number of instructions per message in half compared to two sequential single-message calls. This interleaving approach exploits two sources of parallelism: SIMD data parallelism across the two messages, and reduced data dependencies when the two message streams are independent, which enables additional ILP. The expected speedup over two sequential reference calls is 2×, and any observed speedup that exceeds this bound reflects additional ILP gains.

3.3 Multi-Core Parallelization Using a Thread Pool

Algorithm1 shows the pseudocode for dynamic queue thread pool processing.

Algorithm 1 Dynamic Queue Thread Pool Processing

Require: Message set $\mathcal{M} = \{m_1, \ldots, m_N\}$, batch size $B = 128$
Ensure: Hash value set $\mathcal{H} = \{h_1, \ldots, h_N\}$
Divide $\mathcal{M}$ into $\lceil N/B \rceil$ batches of size B
for each batch $\mathcal{B}_k$ **do**
    Insert $\mathcal{B}_k$ into the queue and signal worker threads
end for
// Worker threads (*running in parallel*)
while queue is not empty **do**
    Acquire mutex, dequeue batch $\mathcal{B}$, release mutex
    for each message m_i in $\mathcal{B}$ **do**
        $h_i \leftarrow$ AVX512_LSH(m_i)
    end for
    Decrement completion counter; signal completion when it reaches 0
end while
Wait for completion signal
return $\mathcal{H}$

The thread pool implementation uses the pthread library to distribute hash computations across all available logical cores. The design consists of three components:

Dynamic Linked-List Queue: Work items are managed as a singly-linked list protected by a mutex. Worker threads enter a waiting state when there are no items to process and wake up immediately via a condition variable when new items are added. A separate completion condition variable notifies the caller when all work items have been processed.

Batch Units: Instead of submitting individual messages one at a time, messages are grouped into batches of size $B = 128$ and inserted into the queue. Each batch is handled as a single work item, reducing synchronization overhead from N to $\lceil N/B \rceil$ for N message submissions. For $N = 10{,}000$ messages, this produces 79 work items, spreading synchronization costs by a factor of 128. The value $B = 128$ was chosen based on

experiments: the throughput difference between $B = 128$ and $B = 256$ is negligible, while smaller batch sizes cause disproportionate overhead for short messages.

Dynamic Load Balancing: Since worker threads independently dequeue items, faster threads automatically process more work items, eliminating the idle time that would arise from static partitioning. The number of worker threads is set automatically based on the number of logical cores in the system, which is 8 on the test platform.

3.4 LEA-128-GCM AVX-512 Optimized Implementation

This section describes the AVX-512 optimized implementation of LEA-128-GCM authenticated encryption. The optimization targets the two bottlenecks of GCM: CTR encryption and GHASH authentication. The KISA LEA-GCM reference processes CTR one block at a time sequentially and handles GHASH one block at a time using SSE instructions, and therefore does not exploit AVX-512-level parallelism. The proposed implementation replaces both components with AVX-512 instructions to improve end-to-end performance.

CTR Encryption: 16-Block Parallelization: As described in Fig.2, GCM's CTR encryption works by encrypting sequentially incrementing counter blocks with a block cipher to produce a keystream, which is then XORed with the plaintext. A counter block is a 128-bit value composed of a 96-bit nonce and a 32-bit counter (big-endian).

Algorithm 2 AVX-512 CTR Mode Encryption

Require: Plaintext P, initial counter c, round keys RK, length ℓ
Ensure: Ciphertext C

while $\ell \geq 256$ **do** ▷ Process 16 blocks at a time
  for $i = 0$ to 15 **do**
    $CTR[i] \leftarrow c + i$ ▷ Big-endian increment
  end for
  $KS \leftarrow \text{AVX512_LEA128}(CTR, RK)$ ▷ Encrypt 16 blocks simultaneously
  for $i = 0$ to 15 **do**
    $C[i] \leftarrow P[i] \oplus KS[i]$
  end for
  $c \leftarrow c + 16; \quad \ell \leftarrow \ell - 256$
end while
while $\ell \geq 16$ **do** ▷ Process remaining blocks
  $KS \leftarrow \text{LEA128}(c, RK)$
  $C \leftarrow P \oplus KS$
  $c \leftarrow c + 1; \quad \ell \leftarrow \ell - 16$
end while
return C

The KISA reference implementation generates counter blocks one at a time and encrypts them with scalar LEA-128, achieving approximately 8

CPB. The proposed implementation processes them in parallel through three steps.
(1) Generate 16 counter blocks sequentially The current counter value is expanded into 16 blocks with values $c, c+1, \ldots, c+15$. Counter increments are performed as big-endian addition on the lower 32 bits.
(2) Encrypt 16 blocks simultaneously with AVX-512 The 16 counter blocks are processed at once using the AVX-512 LEA-128 parallel encryption routine from Choi et al. [5]. This routine packs same-position words from each block into 512-bit registers and applies XAR operations to 16 blocks simultaneously using data parallelism.
(3) XOR keystream with plaintext The keystream is extracted from the 16 encrypted counter blocks and XORed with the plaintext to produce the ciphertext. Remaining bytes below 256 bytes are handled by a single-block path.

GHASH Authentication: VPCLMULQDQ 4-Block Parallelization: GHASH generates an authentication tag by accumulating GF(2^{128}) multiplications over the ciphertext blocks. Sequential GHASH over n blocks is expressed as:

$$S_n = H^n(S_0 \oplus b_0) \oplus H^{n-1}b_1 \oplus \cdots \oplus Hb_{n-1} \tag{1}$$

$H = \mathrm{LEA}(K, 0^{128})$ is the hash subkey. By processing four blocks as a group in Eq. (1), we can pack H^1, H^2, H^3, H^4 into a single 512-bit ZMM register, perform four GF(2^{128}) multiplications simultaneously with VPCLMULQDQ, and XOR the results to obtain the new state. Since Karatsuba-based 128-bit GF multiplication requires four CLMUL operations [7], processing four blocks with PCLMULQDQ (128-bit) requires 16 CLMUL instructions, whereas VPCLMULQDQ (512-bit) performs the same computation with just four instructions.

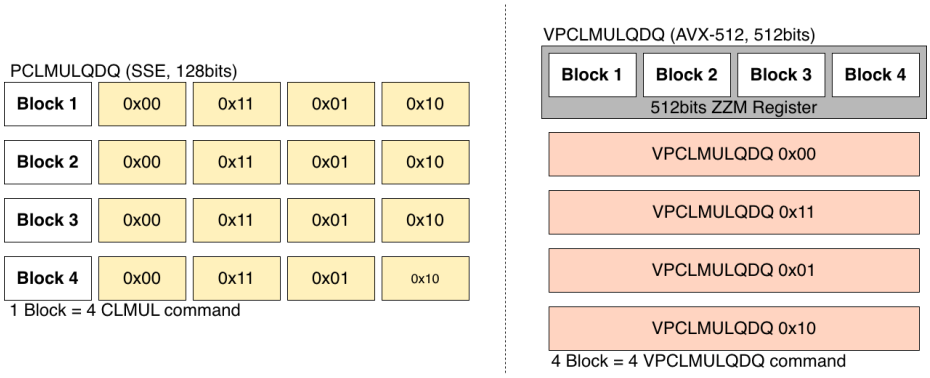

Fig. 3. Comparison of instruction counts between PCLMULQDQ (128-bit) and VPCLMULQDQ (512-bit): 16 instructions reduced to 4 for 4-block GHASH

The VPCLMULQDQ GHASH implementation consists of three steps.

(1) Precompute powers of H at initialization At GCM context initialization, H^1, H^2, H^3, H^4 are computed once and stored in the GCM context. This eliminates the overhead of recomputing powers on every GHASH call and prevents performance degradation on short messages.
(2) VPCLMULQDQ 4-block parallel multiplication Four input blocks and $[H^4, H^3, H^2, H^1]$ are packed into 512-bit ZMM registers respectively, and four VPCLMULQDQ operations are performed according to the Karatsuba decomposition:

$$\begin{aligned} \textbf{lo} &= \texttt{VPCLMULQDQ}(\textbf{b}, \textbf{H}, \texttt{0x00}) \\ \textbf{hi} &= \texttt{VPCLMULQDQ}(\textbf{b}, \textbf{H}, \texttt{0x11}) \\ \textbf{mid} &= \texttt{VPCLMULQDQ}(\textbf{b}, \textbf{H}, \texttt{0x01}) \oplus \texttt{VPCLMULQDQ}(\textbf{b}, \textbf{H}, \texttt{0x10}) \end{aligned}$$

b is the four input blocks packed into a single 512-bit register, and $\mathbf{H} = [H^4 \mid H^3 \mid H^2 \mid H^1]$ is a 512-bit register holding the precomputed powers of H. **lo** and **hi** hold the lower and upper 64-bit multiplication results of each 128-bit lane, **mid** holds the cross terms, and each contains four 128-bit partial products.
The middle terms are distributed across the upper and lower 128-bit lanes to form four 256-bit products.
(3) XOR reduction and GF(2^{128}) reduction The four 256-bit products are XORed lane-by-lane, and then Montgomery reduction is applied with respect to the GCM standard irreducible polynomial $x^{128} + x^7 + x^2 + x + 1$ to update the 128-bit accumulated state. Remaining blocks below 64 bytes are processed by the SSE PCLMULQDQ single-block path.

Correctness Verification: The correctness of the proposed implementation is verified in two steps. First, we confirm that both the LEA-128 scalar implementation and the AVX-512 parallel implementation produce outputs matching the KISA official test vectors for the same key and plaintext. Second, for five plaintext lengths ranging from 16 to 4096 bytes, we verify that the ciphertext and authentication tag produced by the proposed VPCLMULQDQ implementation fully match those of the KISA SSE-based GCM reference.

4 Performance Evaluation

This section quantitatively evaluates the performance of the implementation strategies described in Section 3. Section 4.1 describes the measurement environment and methodology common to all experiments. Sections 4.2–4.4 present the single-message, dual-message, and thread pool results for LSH, respectively. Section 4.5 compares the LEA-GCM optimization results against the KISA reference, and Section 4.6 provides an overall discussion of the observed performance-limiting factors.

4.1 Experimental Setup

Table 1 summarizes the experimental environment.

Table 1. Experimental Environment

Item	Value
Processor	Intel Core i7-1165G7 (Tiger Lake, 11th Gen)
Logical cores	8
Base frequency	2.8 GHz (Turbo Boost disabled)
Operating system	Ubuntu Linux
Compiler	GCC 13.3.0
Compiler flags	`-O3 -mavx512f -mavx512vl -mavx512dq -mavx512bw`
Measurement tools	`clock_gettime(CLOCK_MONOTONIC)`, `rdtsc`
Trials per configuration	30
Confidence interval	95%
CPU affinity	Pinned to core 0 (single-core benchmarks)
Frequency governor	Performance mode

Two performance metrics are reported. *Throughput* (MiB/s) is computed as $(S \times N)/(T \times 2^{20})$, where S is the input size in bytes, N is the number of repetitions, and T is the time in seconds. *Cycles per byte* (CPB) is computed as $C/(S \times N)$, where C is the cycle count. Thirty independent trials are performed for each configuration, and the mean, standard deviation (SD), and 95% confidence interval (CI) are reported. Warm-up iterations are run before all measurements to ensure thermal and cache stability.

4.2 Single-Message Performance

Table 2 presents the single-message results. The baseline is the KISA AVX2-based reference implementation, and the proposed implementation uses AVX-512.

The AVX-512 implementation achieves an average speedup of 1.16× for the LSH-256 family and a higher average speedup of 1.61× for the LSH-512 family.

The performance difference between the two subfamilies is explained by their structural differences. The AVX2 reference requires two 256-bit operations to process the same state, so AVX-512 directly provides a 2× reduction in the number of vector instructions. In contrast, the LSH-256 reference is already efficiently vectorized with AVX2, leaving less room for AVX-512 to improve. The CPB values support this analysis: the LSH-512 CPB drops from approximately 5.1 (reference) to 3.2 (AVX-512), while the LSH-256 CPB decreases more gradually from approximately 5.0 to 4.4.

4.3 Dual-Message Performance

Table 3 presents the dual-message results for LSH-256 variants, comparing two sequential reference calls against a single AVX-512 dual-message call.

All eight configurations exceed the theoretical linear bound of 2.00×, with speedups ranging from 2.23× to 2.31×. The portion exceeding 2.00×

Table 2. Single-message performance: AVX2 reference vs. AVX-512 (runs = 30, repetitions = 200,000). Values are mean ± 95% CI. (*** = ours**).

		AVX2 (reference)		***AVX-512**		
Mode	Input (B)	(MiB/s)	(CPB)	(MiB/s)	(CPB)	Speedup
LSH-256-224	64	331.5±0.4	8.06±0.01	386.9±0.7	6.91±0.01	1.17×
	256	503.4±5.5	5.32±0.07	592.1±0.6	4.51±0.00	1.18×
	1024	699.4±0.3	3.82±0.00	799.8±0.6	3.34±0.00	1.14×
	4096	772.0±1.5	3.46±0.01	876.3±0.3	3.05±0.00	1.14×
LSH-256-256	64	336.7±0.6	7.94±0.01	401.7±0.2	6.66±0.00	1.19×
	256	505.9±0.3	5.28±0.00	593.4±0.5	4.51±0.01	1.17×
	1024	699.2±0.6	3.82±0.01	798.6±3.3	3.35±0.04	1.14×
	4096	772.7±0.8	3.46±0.00	876.0±1.0	3.05±0.01	1.13×
LSH-512-224	64	236.5±0.3	11.30±0.01	377.9±0.2	7.07±0.00	1.60×
	256	530.5±0.5	5.04±0.00	899.6±0.3	2.97±0.00	1.70×
	1024	910.6±4.6	2.94±0.02	1456.9±1.1	1.84±0.00	1.60×
	4096	1111.9±1.7	2.40±0.00	1723.4±1.0	1.55±0.00	1.55×
LSH-512-256	64	240.3±4.5	11.16±0.28	389.9±0.2	6.86±0.00	1.62×
	256	538.3±0.4	4.97±0.01	896.8±1.4	2.98±0.01	1.67×
	1024	917.9±0.9	2.91±0.00	1456.0±0.8	1.84±0.00	1.59×
	4096	1114.9±0.3	2.40±0.00	1719.2±5.7	1.56±0.01	1.54×
LSH-512-384	64	236.7±0.1	11.29±0.01	376.9±0.6	7.09±0.01	1.59×
	256	530.9±0.3	5.04±0.00	898.9±1.4	2.97±0.01	1.69×
	1024	912.6±0.7	2.93±0.01	1457.2±1.0	1.83±0.00	1.60×
	4096	1112.8±0.3	2.40±0.00	1719.4±6.0	1.55±0.01	1.55×
LSH-512-512	64	242.4±0.5	11.03±0.02	389.0±1.8	6.87±0.09	1.60×
	256	538.0±0.4	4.97±0.00	899.0±0.9	2.97±0.01	1.67×
	1024	918.2±0.5	2.91±0.00	1452.0±11.1	1.84±0.04	1.58×
	4096	1114.5±0.4	2.40±0.00	1721.8±4.2	1.55±0.01	1.54×

is attributed to ILP: since the two message streams are independent, the out-of-order execution units can overlap instructions from both streams, effectively shortening the critical path. The 1024 B and 4096 B inputs show slightly higher speedups than the smaller 64 B inputs, which is consistent with the ILP benefit increasing as it is spread over more computation rounds.

4.4 Thread Pool Performance

Table 4 presents the thread pool results, comparing single-threaded AVX-512 sequential execution against 8-thread pool execution for $N_{\text{msg}} = 10{,}000$ messages per measurement.

Table 3. Dual-message performance: 2×AVX2 reference vs. AVX-512 dual-message (runs = 30, repetitions = 200,000). Throughput reflects 2× data; values are mean ± 95% CI (MiB/s). (*** = ours**).

Mode	Input (B)	2×Reference	***AVX-512 Dual**	Speedup
LSH-256-224	64	331.6±0.1	737.9±1.2	2.23×
	256	505.1±0.2	1142.0±13.1	2.26×
	1024	698.3±1.5	1602.8±1.0	2.30×
	4096	772.7±0.5	1774.8±3.9	2.30×
LSH-256-256	64	336.0±0.8	760.2±0.4	2.26×
	256	505.4±0.2	1160.6±2.2	2.30×
	1024	698.4±1.3	1608.8±0.8	2.30×
	4096	772.3±0.8	1780.2±0.4	2.31×

The thread pool achieves speedups of 3.50× to 5.12× across all configurations. The overall average speedup is 4.25× with 8 threads, which falls short of the theoretical maximum of 8.00×. Two systematic trends are observed. First, speedup increases with input size: 64 B inputs yield speedups of 3.55–4.00×, while 4096 B inputs reach 4.94–5.12×. This reflects the decreasing relative contribution of thread synchronization overhead as per-message computation time grows. Second, the standard deviation of the thread pool results is considerably higher than that of the single-core benchmarks, reflecting the non-deterministic nature of multi-core scheduling.

4.5 LEA-GCM Performance

The performance evaluation was conducted using the 128-bit key of LEA. Table 5 compares LEA-GCM authenticated encryption performance across three implementations: a pure software GHASH implementation (SW), an SSE PCLMULQDQ-based implementation (SSE), and the proposed implementation (AVX) combining AVX-512 CTR (16-block) and VPCLMULQDQ GHASH (4-block). The performance metric is CPB (cycles per byte) measured on plaintext bytes; the AAD (13 bytes) and authentication tag (16 bytes) are not included in the denominator. Speedup is computed as SSE CPB divided by AVX CPB.

The proposed implementation achieves a 3.26× improvement over the SSE-based implementation and a 12.1× improvement over the software implementation for 4096-byte inputs. Figure 4 visually compares CPB by message size.

The performance improvement is explained by the complementary contributions of two optimizations.

CTR AVX-512 contribution: Replacing scalar CTR encryption (approximately 8 CPB) with AVX-512 16-block parallel processing significantly reduces the CTR bottleneck, which accounted for roughly 8 CPB

Table 4. Thread pool performance: AVX-512 sequential vs. 8-thread dynamic queue pool (runs = 30, N_{msg} = 10,000, batch = 128). Values are mean ± 95% CI (MiB/s). (*** = ours**).

Mode	Input (B)	Sequential	***Thread Pool**	Speedup
LSH-256-224	64	370.9±16.2	1318.3±89.6	3.55×
	256	591.4±1.5	2205.8±113.0	3.73×
	1024	791.1±17.5	3071.5±136.3	3.88×
	4096	876.6±0.2	4442.6±76.6	5.07×
LSH-256-256	64	400.6±0.3	1443.3±53.8	3.60×
	256	592.2±1.3	2562.7±47.7	4.33×
	1024	800.2±0.2	3811.3±46.6	4.76×
	4096	876.2±1.1	4480.0±69.8	5.11×
LSH-512-224	64	376.0±0.3	1429.9±43.2	3.80×
	256	895.8±2.7	3542.4±103.3	3.95×
	1024	1454.7±1.1	5090.9±171.8	3.50×
	4096	1721.4±4.1	6787.2±320.7	3.94×
LSH-512-256	64	371.8±16.1	1488.7±62.7	4.00×
	256	897.2±0.5	3550.2±74.9	3.96×
	1024	1455.7±0.5	6641.4±90.0	4.56×
	4096	1723.2±0.8	8707.4±126.1	5.05×
LSH-512-384	64	375.2±0.5	1474.4±41.5	3.93×
	256	897.5±0.8	3596.0±80.0	4.01×
	1024	1456.3±0.5	6662.8±78.1	4.58×
	4096	1723.8±0.3	8831.3±129.2	5.12×
LSH-512-512	64	388.8±0.6	1490.4±29.3	3.83×
	256	897.6±0.4	3501.4±78.3	3.90×
	1024	1456.1±0.5	6720.2±86.6	4.62×
	4096	1723.6±0.8	8811.7±139.6	5.11×

out of the total 9.89 CPB in the SSE-based implementation. This is the primary contributor to the performance improvement.

VPCLMULQDQ GHASH contribution: Replacing SSE-based GHASH (approximately 2 CPB) with VPCLMULQDQ 4-block parallel processing further reduces the GHASH overhead. Replacing only CTR with AVX-512 yields 4.02 CPB, and additionally replacing GHASH with VPCLMULQDQ yields 3.03 CPB, giving an additional 1.33× improvement.

Message length dependency: The lower speedup at 64 bytes (1.08×) compared to larger inputs is due to the relatively large GCM initialization overhead for short messages. For inputs of 256 bytes and above, the

Table 5. LEA-GCM authenticated encryption performance comparison (runs = 30, repetitions = 200,000). Values are mean ± 95% CI. (*** = ours**).

Input (B)	SW (CPB)	SSE (CPB)	***AVX (CPB)**	Speedup
64	53.83±0.26	14.28±0.03	13.23±0.02	1.08×
256	40.61±0.02	10.78±0.01	4.40±0.00	2.45×
1024	37.40±0.03	10.00±0.02	3.30±0.00	3.03×
4096	36.65±0.01	9.89±0.01	3.03±0.00	3.26×

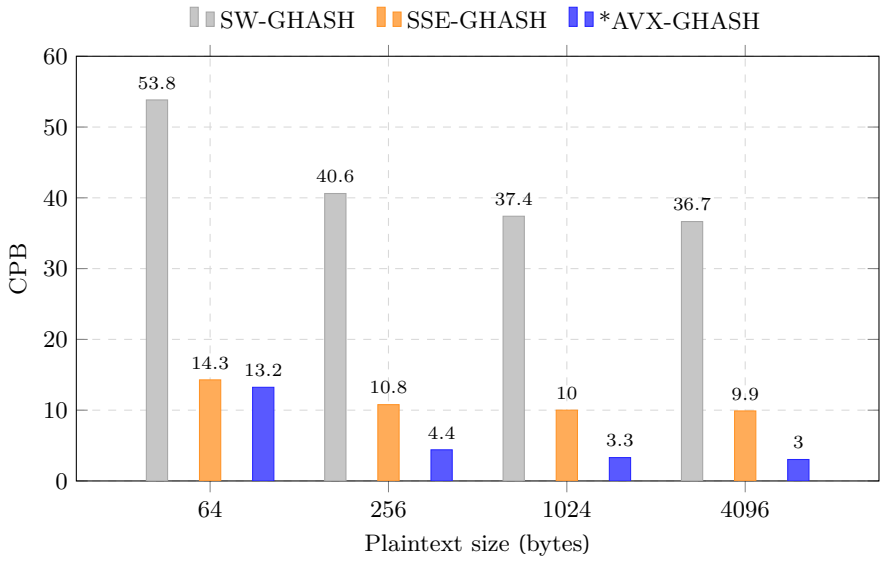

Fig. 4. LEA-128-GCM authenticated encryption performance comparison (CPB, lower is better)

4-block processing benefit of VPCLMULQDQ consistently exceeds the overhead, and the speedup shows a stable increasing trend.

4.6 Discussion

The results collectively answer the research questions and reveal two performance-limiting factors: AVX-512 frequency throttling and memory bandwidth contention.

Intel's AVX-512 license transition lowers the processor clock when 512-bit operations are active. When 8 cores execute AVX-512 instructions simultaneously, the package frequency drops much more than during single-core execution, compressing the effective per-core performance available in the thread pool. This is the main reason why the observed thread pool speedups fall well short of the theoretical maximum.

In addition, when 8 cores access memory simultaneously, cache and bus bandwidth become shared resources. Hash computation is a streaming workload: each message is read once, processed, and discarded, creating continuous memory pressure. The highest speedups observed for large messages (4096 B) suggest that larger inputs more effectively spread both synchronization overhead and memory latency.

5 Conclusion

This paper proposed AVX-512-based optimized implementations of two Korean standard cryptographic algorithms—the LSH hash function and LEA-GCM authenticated encryption—and demonstrated that the proposed strategies lead to measurable performance improvements.

For LSH optimization, three strategies were applied: single-message processing using AVX-512 ZMM registers, dual-message simultaneous processing based on register interleaving, and multi-core scaling through a dynamic queue thread pool. The single-message implementation achieved 1.16× throughput improvement for LSH-256 and 1.61× for LSH-512 over the KISA AVX2 reference. Dual-message interleaving achieved an average superlinear speedup of 2.28×, exceeding the theoretical linear bound of 2× through instruction-level parallelism (ILP). The thread pool achieved scalability of 3.50–5.12× in an 8-core environment.

For LEA-GCM optimization, 16-block AVX-512 parallelization of CTR encryption was combined with VPCLMULQDQ 4-block parallelization of GHASH authentication. This strategy achieved a 3.26× authenticated encryption throughput improvement over the KISA SSE-based implementation and a 12.1× improvement over the pure software implementation for 4096-byte inputs. Correctness was verified by matching the KISA official test vectors.

These results provide a systematic basis for the AVX-512 optimization potential of Korean standard cryptographic algorithms. By demonstrating that performance improvements using the AVX-512 instruction set are achievable for both LSH hash function and LEA-GCM authenticated encryption, this work establishes a practical performance baseline for the adoption of Korean standard cryptography.

Several directions remain for future work. First, since the experiments were conducted on a single Intel Tiger Lake platform, repeating the experiments on other AVX-512-capable platforms would establish the generalizability of the speedup results. Second, applying VPCLMULQDQ parallelization of GHASH beyond 8 blocks and CTR-GHASH pipelining in LEA-GCM is expected to yield further improvements for large messages. Third, an AVX-512 implementation of a combined authenticated hash structure using LSH and LEA-GCM could be developed into a comprehensive study evaluating the performance of the Korean standard cryptographic suite as a whole.

References

1. Bellizia, D., Berti, F., Bronchain, O., Cassiers, G., Duval, S., Guo, C., Leander, G., Leurent, G., Levi, I., Momin, C., et al.: Spook: Sponge-based leakage-resistant authenticated encryption with a masked tweakable block cipher. IACR Transactions on Symmetric Cryptology **2020**(S1), 295–349 (2020)
2. Chan-hee, P., Hae-yong, K., Jang-hyun, J., Jin-jae, L., Ho-won, K.: To Design and Work with Cryptographic Modules for KCMVP Authentication. Proceedings of the Korea Institute of Communication Sciences Conference (2019)
3. Cheng, H., Fotiadis, G., Großschädl, J., Ryan, P.Y.: Highly vectorized SIKE for AVX-512. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 41–68 (2022)
4. Choi, Hojin and Choi, SeongJun and Seo, SeogChung: Parallel implementation of lightweight secure hash algorithm on CPU and GPU environments. Electronics **13**(5), 896 (2024)
5. Choi, Yongryeol and Choi, Hojin and Seo, Seog Chung: AVX512crypto: parallel implementations of Korean block ciphers using AVX-512. IEEE Access **11**, 55094–55106 (2023)
6. Dworkin, M.J.: Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC (2007)
7. Gueron, S., Kounavis, M.E.: Intel® carry-less multiplication instruction and its usage for computing the GCM mode. White Paper **10** (2010)
8. Hong, Deukjo and Lee, Jung-Keun and Kim, Dong-Chan and Kwon, Daesung and Ryu, Kwon Ho and Lee, Dong-Geon: LEA: A 128-bit block cipher for fast encryption on common processors. In: international workshop on information security applications. pp. 3–27. Springer (2013)
9. Intel Corporation: Intel® Architecture Instruction Set Extensions and Future Features Programming Reference. Tech. Rep. 319433-047, Intel Corporation (December 2022), `https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html`
10. Intel Corporation: Intel® 64 and IA-32 Architectures Software Developer's Manuals (2026), `https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html`, Accessed: 2026-05-11
11. Käsper, E., Schwabe, P.: Faster and timing-attack resistant AES-GCM. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 1–17. Springer (2009)
12. Kim, Dong-Chan and Hong, Deukjo and Lee, Jung-Keun and Kim, Woo-Hwan and Kwon, Daesung: LSH: A new fast secure hash function family. In: International Conference on Information Security and Cryptology. pp. 286–313. Springer (2014)
13. Korea Internet & Security Agency (KISA): Source Code and Manual for GCM/CCM Operation Modes of Domestic Block Ciphers (LEA, ARIA, SEED). `https://seed.kisa.or.kr/kisa/Board/20/detailView.do` (2021), Accessed: 2026-05-11

14. McGrew, D., Viega, J.: The Galois/counter mode of operation (GCM). submission to NIST Modes of Operation Process **20**, 0278–0070 (2004)
15. National Security Research Institute: Specification of Lightweight Secure Hash Algorithm LSH. Tech. rep., National Security Research Institute (NSR), Daejeon, Republic of Korea (2019), Available online
16. Rescorla, Eric: The transport layer security (TLS) protocol version 1.3 (2018)
17. Seo, Hwajeong and Park, Taehwan and Heo, Shinwook and Seo, Gyuwon and Bae, Bongjin and Hu, Zhi and Zhou, Lu and Nogami, Yasuyuki and Zhu, Youwen and Kim, Howon: Parallel implementations of LEA, revisited. In: International Workshop on Information Security Applications. pp. 318–330. Springer (2016)
18. Takahashi, D.: An implementation of parallel number-theoretic transform using intel AVX-512 instructions. In: International Workshop on Computer Algebra in Scientific Computing. pp. 318–332. Springer (2022)
19. Wang, Z., Chen, H., Cai, W.: A hybrid CPU/GPU scheme for optimizing ChaCha20 stream cipher. In: 2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom). pp. 1171–1178. IEEE (2021)