


Article

Optimization of SM2 Algorithm Based on Polynomial Segmentation and Parallel Computing

Hongyu Zhu ^{1,2}, Ding Li ^{3,4,*}, Yizhen Sun ^{1,2}, Qian Chen ⁵, Zheng Tian ^{1,2} and Yubo Song ^{3,4} 

¹ State Grid Hunan Electric Power Company Limited Information and Communication Company, Changsha 410004, China; zhuhy17@hn.sgcc.com.cn (H.Z.); sunyz2@hn.sgcc.com.cn (Y.S.); tianz5@hn.sgcc.com.cn (Z.T.)

² Hunan Key Laboratory for Internet of Things in Electricity, Changsha 410004, China

³ School of Cyber Science and Engineering, Southeast University, Nanjing 211189, China; songyubo@seu.edu.cn

⁴ Purple Mountain Laboratories, Nanjing 211111, China

⁵ State Grid Hunan Electric Power Co., Ltd., Changsha 410007, China; chenq31@hn.sgcc.com.cn

* Correspondence: ld1ng@seu.edu.cn

Abstract: The SM2 public key cryptographic algorithm is widely utilized for secure communication and data protection due to its strong security and compact key size. However, the intensive large integer operations it requires pose significant computational challenges, which can limit the performance of Internet of Things (IoT) terminal devices. This paper introduces an optimized implementation of the SM2 algorithm specifically designed for IoT contexts. By segmenting large integers as polynomials within a modified Montgomery modular multiplication algorithm, the proposed method enables parallel modular multiplication and reduction, thus addressing storage constraints and reducing computational redundancy. For scalar multiplication, a Co-Z Montgomery ladder algorithm is employed alongside Single Instruction Multiple Data (SIMD) instructions to enhance parallelism, significantly improving efficiency. Experimental results demonstrate that the proposed scheme reduces the computation time for the SM2 algorithm's digital signature by approximately 20% and enhances data encryption and decryption efficiency by about 15% over existing methods, marking a substantial performance gain for IoT applications.

Keywords: Montgomery modular multiplication; SIMD; Montgomery ladder; parallel processing; elliptic curve cryptography; modular arithmetic optimization; efficiency improvement



Citation: Zhu, H.; Li, D.; Sun, Y.; Chen, Q.; Tian, Z.; Song, Y.

Optimization of SM2 Algorithm Based on Polynomial Segmentation and Parallel Computing. *Electronics* **2024**, *13*, 4661. <https://doi.org/10.3390/electronics13234661>

Academic Editor: Carlo Mastroianni

Received: 1 October 2024

Revised: 18 November 2024

Accepted: 19 November 2024

Published: 26 November 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Elliptic curve cryptography (ECC), compared with the RSA cryptographic algorithm, offers better computational efficiency and requires less key storage space. Consequently, ECC has become a crucial alternative to RSA. Major countries worldwide have established ECC standards and are promoting their use to safeguard information security.

ECC is applied in various scenarios. For instance, the U.S. government uses ECC as a fundamental cryptographic component to ensure data security in the Tor project [1]. In the foundational technology of Bitcoin [2] blockchain, Nakamoto employs ECC as the authentication mechanism. However, international ECC standards are not completely secure. Internal memos leaked by former NSA employee Edward Joseph Snowden [3] reveal that the NSA left a backdoor in the NIST's ECC technical standards, where the pseudo-random number generator Dual_EC_DRBG [4] can have its output predicted under certain conditions. Therefore, it is crucial to adopt cryptographic algorithms independently developed by our country for critical information systems.

In 2010, the National Cryptography Administration of China introduced the SM2 public key cryptography algorithm based on ECC, aimed at replacing the RSA algorithm. As a national cryptographic algorithm, SM2 has been applied in various commercial cryptographic products, such as electronic certification systems and key management

systems. Additionally, due to its high security and short key length, the SM2 algorithm is widely used for key agreement and data encryption transmission among IoT devices. IoT devices have brought significant convenience to our production and daily life, from simple smart home controls to complex industrial automation equipment. However, these devices face numerous limitations in computational capacity and storage space, placing higher demands on the performance of the SM2 algorithm. Specifically, because it relies on operations over elliptic curve groups, the computational load is high, its structure is complex, and it suffers from low operational efficiency and high resource consumption on resource-limited IoT terminals.

Recently, researchers have proposed various solutions for the SM2 algorithm tailored to different application scenarios, primarily focusing on applications [5–9], with less emphasis on optimization implementation. Since the SM2 algorithm has not been available for long, it does not adapt well to some existing ECC computational algorithms, hindering its broader adoption. Discussing the integration of these algorithms with the SM2 computational algorithm is crucial for enhancing the computational efficiency of the SM2 algorithm.

The main contributions of this paper are as follows:

1. Efficient elliptic curve group computation: we utilize an improved Montgomery algorithm based on polynomial expansion, where large integers are expanded into polynomials to interleave modular multiplication and modular reduction operations. This approach achieves efficient and stable modular multiplication. Additionally, the Montgomery ladder algorithm based on Co-Z operations reduces redundant calculations in elliptic curve scalar multiplication, further enhancing the efficiency of point addition and doubling.
2. Effective use of parallel computing: we introduce SIMD instruction set optimization to enable parallel processing of multiple data. In particular, the combination of SIMD technology with Montgomery point multiplication significantly improves performance.
3. Comprehensive implementation of the SM2 national cryptographic algorithm: we fully implement the SM2 public key encryption algorithm and digital signature algorithm, ensuring wide applicability in practical applications. Our design is compared with other public algorithms such as the SM2 signature algorithm based on OpenSSL 1.1.1, the SM2 encryption and decryption algorithm, ECDSA (secp256k1), and ECIES (secp256k1). Metrics such as runtime and throughput are used as evaluation criteria.

2. Related Work

Numerous scholars worldwide have extensively researched the ECC algorithm. The optimization of field operations primarily targets computationally intensive modular multiplication. In 1985, mathematician Montgomery proposed Montgomery multiplication [10], significantly speeding up large integer modular multiplication. Building on this, Koc et al. [11] integrated the steps of the Montgomery reduction algorithm, proposing five optimized versions that further improved efficiency. In 1987, Montgomery introduced a point multiplication algorithm over binary extension fields [12], which only requires processing the X coordinate during point multiplication, thus reducing space requirements. In 1999, literature [13] extended the Montgomery point multiplication algorithm to LD coordinate representation, reducing the need for inversion operations in binary fields. In 2010, Ref. [14] used a bottom-up approach to optimize the ECC point multiplication algorithm, employing Jacobi coordinates, extended Twisted Edwards coordinates, and the Galbraith–LinScott (GLS) method, achieving about a 30% improvement in computational speed.

Since Intel introduced the SIMD instruction set in 1996, many researchers have used this instruction set to optimize various symmetric cryptographic algorithms. Parallel implementation schemes for the SM4 algorithm [15] and the ZUC algorithm [16] have subsequently been proposed. The key steps of elliptic curve-based public key cryptographic algorithms are large integer operations and elliptic curve operations. Hisil et al. [4], based on the Montgomery ladder point multiplication algorithm [12], designed a four-way vec-

torized Montgomery point multiplication algorithm for Montgomery form elliptic curves, utilizing AVX2 and AVX512 instruction sets. Nath et al. [17] further proposed optimized algorithms for variable base point and fixed-base point scalar multiplication, reducing the number of multiplications in point operations. They implemented a four-way vectorized Montgomery point multiplication algorithm for Curve25519 and Curve448 curves using assembly language and the AVX2 instruction set, significantly reducing the runtime. Cheng et al. [18] further improved the throughput of scalar multiplication on the Curve25519 curve based on the AVX2 instruction set. Huang et al. [19] extended this approach to the SM2 elliptic curve, implementing a two-way parallel large integer operation and scalar multiplication algorithm using the AVX2 instruction set, which reduced the latency of constant-time scalar multiplication on SM2. Gueron et al. [20] developed a highly optimized cryptographic library for the NIST P-256 curve on the x86_64 platform, optimizing the fixed-base scalar multiplication computation process using SIMD instructions, which can be directly extended to OpenSSL. Bernstein et al. [21] proposed techniques for accelerating fixed-base scalar multiplication on the Ed25519 curve, increasing the running speed of the EdDSA scheme. Faz-Hernández et al. [22] designed a precomputation table for the Ed25519 curve using SIMD instructions, further optimizing the performance of fixed-base scalar multiplication.

Additionally, various other optimization efforts have been made. F. Chen et al. [23] proposed a low-cost, high-speed parallel modular multiplication implementation for SM2, executing the two steps of multiplication and reduction in parallel. They improved the classic Karatsuba algorithm, executing it in eight parts, combined with subsequent parallel reduction computations. Wang et al. [24] proposed a modular squaring unit using a fast partial Montgomery reduction algorithm, significantly reducing area. Modular squaring operations can be completed in only four clock cycles, and modular multiplication and squaring can be computed in parallel for high speed. Xingran Li et al. [25] proposed a new parallel efficient algorithm to accelerate scalar multiplication, introducing a new regular halving and addition method, effective with projective coordinates. They compared various algorithms for double addition and halving addition, finally combining the best methods to obtain a new faster parallel algorithm. Mai et al. [26] developed a new SM2 algorithm implementation called Yog-SM2, leveraging the parallel computing capabilities of modern processors, such as the AVX2 instruction set, significantly enhancing execution speed while maintaining security. Hu et al. [27] proposed a high-performance ECC architecture based on a half-word multiplier, where PM operations consist of point addition (PA) and point doubling (PD) operations. They introduced a novel schedule for PA and PD to reduce MM operations and enhance the parallelism of multiplication and fast reduction operations.

3. Preliminaries

3.1. SM2 Public Key Cryptographic Algorithm

The SM2 algorithm is a type of ECC algorithm. It provides higher security strength than 2048-bit RSA and operates faster than RSA. The National Cryptography Administration of China has decided to use the SM2 algorithm to replace RSA after thorough research. The SM2 elliptic curve public key cryptography algorithm was developed and formulated by China, incorporating existing research from both domestic and international sources, and it possesses independent intellectual property rights.

An elliptic curve system over a prime field typically consists of the following parameters:

- (1) The field size p , where p is a prime number greater than 3;
- (2) The elliptic curve equation $y^2 = x^3 + ax + b$, with coefficients a and b ;
- (3) The base point $G(G_x, G_y)$ on the elliptic curve;
- (4) The order n of the base point;
- (5) The cofactor h of the base point.

SM2 digital signature algorithm: the SM2 elliptic curve cryptography algorithm includes digital signature generation and verification, as well as encryption and decryption. Given a message, M , to be signed, the signature, (r, s) , the public key, P , and the private key,

d , the elliptic curve equation over the finite prime field, p , is defined as $y^2 = x^3 + ax + b$ with the base point G . The signing process is as follows. Let Z_A be the hash value of the user A , M be the message to be signed, and d_A be the private key of user A :

- (1) Concatenate Z_A and M , denoted as M' , i.e., $M' = Z_A \parallel M$;
- (2) Compute the hash value of M' , denoted as e , i.e., $e = H_v(M')$;
- (3) Generate a random number $k \in [1, n - 1]$;
- (4) Compute the elliptic curve point $(x_1, y_1) = [k]G$;
- (5) Compute $r = (e + x_1) \bmod n$. If $r = 0$ or $r + k = n$, return to step 3;
- (6) Compute $s = [(1 + d_A)^{-1} \cdot (k - r \cdot d_A)] \bmod n$. If $s = 0$, return to step 3;
- (7) Output the digital signature (r, s) for the message M .

SM2 encryption algorithm: given the message M to be sent as a bit string of length $klen$, the SM2 encryption algorithm operates as follows:

- (1) Generate a random number $k \in [1, n - 1]$;
- (2) Compute the elliptic curve point $C_1 = (x_1, y_1) = [k]G$;
- (3) Compute the elliptic curve point $S = [h]P_B$. If $S = O$, report an error and exit;
- (4) Compute the elliptic curve point $(x_2, y_2) = [k]P_B$;
- (5) Compute $t = KDF(x_2 \parallel y_2, klen)$. If t is an all-zero bit string, return to step 1;
- (6) Compute $C_2 = M \oplus t$;
- (7) Compute $C_3 = H_v(x_2 \parallel M \parallel y_2)$;
- (8) Output the ciphertext $C = C_1 \parallel C_2 \parallel C_3$.

3.2. Montgomery Modular Multiplication

The classic Montgomery modular multiplication algorithm leverages the properties of residue systems to transform standard modular operations into shift and addition operations. The algorithm computes $S = A \times B \times R^{-1} \bmod M$. Select large integers A and B such that $0 \leq A, B < M$ and $\text{GCD}(M, R) = 1$, and choose a base, R , where $R > M$. Parameters R^{-1} and M' are selected to satisfy $0 < R^{-1} < M$, $0 < M' < R$, and $R \times R^{-1} - M' \times M = 1$, ensuring the following conditions hold:

$$\begin{aligned} R \times R^{-1} &\equiv 1 \pmod{M} \\ M' &\equiv -M^{-1} \pmod{R} \end{aligned} \quad (1)$$

The classic Montgomery algorithm consists of four main steps:

1. Multiply the large integers A and B to obtain the product T : $T = A \times B$;
2. Take the lower n bits of T , multiply by M' , and again take the lower n bits to obtain the quotient q : $q = (T \bmod R) \times M' \bmod R$;
3. Multiply q by the modulus M , add T , and take the higher n bits to obtain S : $S = \frac{T + q \times M}{R}$;
4. If $S > M$, return $S - M$; otherwise, return S .

By analyzing the classic Montgomery algorithm and combining it with Equation (1), we obtain:

$$q \times M \equiv -T \pmod{R} \quad (2)$$

and

$$S \times R \equiv T \pmod{R}, \text{ which implies } S \equiv T \times R^{-1} \pmod{M} \quad (3)$$

Since $0 \leq T + q \times M < 2 \times M \times R$, an additional step, as shown in step 4, is necessary to obtain the final result. The classic Montgomery algorithm involves two n -bit large integer multiplications and two $2n$ -bit large integer additions, making the logic operations more complex and time-consuming with wider bit-widths.

3.3. Montgomery Ladder Algorithm

The Montgomery ladder algorithm is designed for computing non-fixed-point scalar multiplication on Montgomery curves. This algorithm focuses only on the X coordinate,

eliminating many Y coordinate operations. In the main loop, each iteration performs a point addition and a point doubling operation, with a time complexity of only $5M + 4S$, making it significantly more efficient than other elliptic curve algorithms. The algorithm's implementation involves conditional swaps to resist side-channel attacks, making it highly suitable for ECDH protocols requiring efficient non-fixed-point scalar multiplication.

As shown in Algorithm 1, the Montgomery point multiplication algorithm is presented. It starts by computing the initial values $R_0 = G$ and $R_1 = 2G$ based on the base point G . It then performs point addition and doubling operations on R_0 and R_1 according to the binary encoding of the integer k . The loop computes the final point multiplication result based on the binary digits of k .

Algorithm 1 Montgomery Point Multiplication

```

1: Input:  $k = (k_{n-1}, \dots, k_0)$ , point  $G$ 
2: Output:  $Q = kG$ 
3: Initialize  $R_0 = G, R_1 = 2G, i = n - 2$ 
4: while  $i \geq 0$  do
5:   if  $k_i == 0$  then
6:      $R_1 = R_0 + R_1, R_0 = 2R_0$ 
7:   else
8:      $R_0 = R_0 + R_1, R_1 = 2R_1$ 
9:   end if
10:   $i = i - 1$ 
11: end while
12:  $Q = R_0$ 

```

4. Methods

4.1. Polynomial Expansion Cross Montgomery Modular Multiplication Algorithm

This paper presents an improved version of the classical Montgomery modular multiplication algorithm, resulting in a polynomial expansion cross Montgomery modular multiplication algorithm. This algorithm consists of two main loops, referred to as Loop 1 and Loop 2. Loop 1 calculates the intermediate parameter, q_i , used in modular reduction, and Loop 2 performs multiple iterations to complete both the modular multiplication and reduction operations, with Loop 3 embedded within it to handle individual iterations of these operations.

In this algorithm, the input parameters—the modulus, multiplicand, and multiplier—are represented as polynomials:

$$M = \sum_{j=0}^{k_2-1} m_j \times (r_A)^j, \quad A = \sum_{j=0}^{k_2-1} m_j \times (d_A)^j, \quad y = r_A = 2^y, \quad k_2 = \frac{n}{y} \quad (4)$$

$$A = \sum_{j=0}^{k_2-1} a_j \times (r_A)^j = \sum_{j=0}^{k_2-1} a_j \times (d_A)^j, \quad y = r_A = 2^y, \quad 0 \leq A \leq 2M, \quad k_2 = \frac{n}{y} \quad (5)$$

$$B = \sum_{i=0}^{k_1-1} b_i \times (r_B)^i = \sum_{i=0}^{k_1-1} b_i \times (d_B)^i, \quad x = r_B = 2^x, \quad 0 \leq B \leq 2M, \quad 4M < 2^{k_1x} \quad (6)$$

The input parameter is M'_0 , which represents the lower x bits of the modular multiplicative inverse of M (denoted M'), $S_0 = 0$. The output parameter is S .

The algorithm proceeds as follows:

Step 1: Sequentially take the low x bits of B (denoted b_i) and multiply by the low x bits of A (denoted $A[0]$). Add the low x bits of the intermediate result S_i (denoted $S_i[0]$). After the multiply–accumulate operation, retain the low x bits and multiply by M'_0 . Finally, take the low x bits to obtain the parameter q_i . As shown in Algorithm 2, the iterative

calculation of the intermediate parameter q_i is performed to ensure computational accuracy and efficiency.

Algorithm 2 Iterative Calculation of Intermediate Parameter q_i

```

1: for  $i = 0$  to  $k_1 - 1$  do                                     ▷ Loop 1
2:   if  $i == 0$  then                                           ▷ Initialize
3:      $S_i = 0$ 
4:      $q_i = (((S_i[0] + A[0] \times b_i) \bmod x) \times M'_0) \bmod x$ 
5:   end if
6: end for

```

Step 2: Utilize a nested loop structure, consisting of Loop 2 and Loop 3. In each iteration of the inner loop (Loop 3), sequentially take the y bits of A (denoted a_j) and multiply them by the x bits of B (denoted b_i , where i ranges from 0 to 30). Accumulate the result with the y bits of the previous iteration S_i (denoted $S_{i,j}$) and the carry $C_{AB_{i+1,j}}$. This yields the modular multiplication result. Subsequently, multiply q_i by the y bits of the modulus M (denoted m_j) and add the carry $C_{qm_{i+1,j}}$ and the accumulated result $S_{AB_{i+1,j+1}}$ to complete the modular reduction operation, as shown in Algorithm 3.

Algorithm 3 Nested Modular Multiplication and Reduction Algorithm

```

1: for  $i = 0$  to  $k_1 - 1$  do                                     ▷ Loop 2
2:   if  $i == 0$  then                                           ▷ Initialize
3:      $S_i = 0$ 
4:   end if
5:   for  $j = 0$  to  $k_2 - 1$  do                                     ▷ Loop 3
6:     if  $j == 0$  then                                           ▷ Initialize
7:        $C_{AB_{i+1,j}} = 0$ 
8:     end if
9:      $(C_{AB_{i+1,j+1}}, S_{AB_{i+1,j+1}}) = b_i \times a_j + S_{i,j} + C_{AB_{i+1,j}}$ 
10:     $(C_{qm_{i+1,j+1}}, S_{qm_{i+1,j+1}}) = q_i \times m_j + S_{AB_{i+1,j+1}} + C_{qm_{i+1,j}}$ 
11:   end for
12:    $S_{i+1} = \frac{S_{qm_{i+1,j+1}}}{x}$ 
13: end for

```

Step 3: The algorithm expands the range of input data, thereby eliminating the need for the subtraction operation used in the classical Montgomery modular multiplication algorithm. The computation concludes with the output result S .

4.1.1. Polynomial Expansion

Assuming the n -bit large integer B is represented with base r_B , $B = (b_{k_1-1}, b_{k_1-2}, \dots, b_1, b_0)r_B$, with $r_B = 2^x$. Each part is represented by b_i , where b_i ranges from 0 to x , and is calculated according to Equation (4). Here, i ranges from 0 to $k_1 - 1$, and the integers x and k_1 satisfy $4M < 2^{(k_1 x)}$, resulting in the k_1 -bit large integer B (Equation (5)):

$$b_i = (B \gg (x \times i)) \bmod r_B \quad (7)$$

$$B = \sum_{i=0}^{k_1-1} b_i \times (r_B)^i \quad (8)$$

Additionally, the k_1 -bit large integer B can also be expanded as a polynomial of degree $k_1 - 1$, as shown in Equation (6), where $d_B = r_B$:

$$B(d_B) = \sum_{i=0}^{k_1-1} b_i \times (d_B)^i \quad (9)$$

Similarly, assuming the n -bit large integer A is represented with base r_A , $A = (a_{k_2-1}, a_{k_2-2}, \dots, a_1, a_0)r_A$, with $r_A = 2^y$. Each part is represented by a_j , where a_j ranges from 0 to y , and is calculated according to Equation (7). Here, j ranges from 0 to $k_2 - 1$, and $k_2 = \left\lceil \frac{n}{y} \right\rceil$, resulting in A :

$$a_j = (A \gg (d_A \times j)) \bmod r_A \quad (10)$$

$$A = \sum_{j=0}^{k_2-1} a_j \times (r_A)^j \quad (11)$$

Moreover, the k_2 -bit large integer A can also be expanded as a polynomial of degree $k_2 - 1$, as shown in Equation (9), where $d_A = r_A$:

$$A(d_A) = \sum_{j=0}^{k_2-1} a_j \times (d_A)^j \quad (12)$$

$$T = A \times B = A(d_A) \times B(d_B) = \sum_{i=0}^{k_1-1} \left\{ b_i \times (d_B)^i \times \left(\sum_{j=0}^{k_2-1} (a_j \times (d_A)^j) \right) \right\} \quad (13)$$

If the input data are the n -bit multiplicand A and the multiplier B , the computation process will produce an intermediate result of up to $2n$ bits. By expanding the input data as polynomials and performing multi-precision calculations, each operation only involves parts of A and B (i.e., a_j and b_i), reducing the intermediate result to $(x + y)$ bits. In hardware implementation, this multiplier can be reused, and its size will affect the circuit's area, making this operation beneficial for saving area costs.

4.1.2. Cross Execution of Modular Multiplication and Modular Reduction

This paper illustrates the use of the polynomial expansion modular multiplication algorithm, which involves multiplication and modular reduction operations, by combining it with Step 2 of the polynomial expansion cross Montgomery modular multiplication algorithm. Step 2 primarily consists of Loop 2 and Loop 3, with Loop 3 nested within Loop 2. Loop 3 executes the operations described in Equations (11) and (12), achieving modular multiplication and modular reduction, respectively. The computation within Loop 2 is as shown in Equation (13):

$$S_{AB_{i+1}} = b_i \times \left(\sum_{j=0}^{k_2-1} a_j \times (d_A)^j \right) = b_i \times A \quad (14)$$

$$S_{qm_{i+1}} = q_i \times \left(\sum_{j=0}^{k_2-1} m_j \times (d_B)^j \right) + S_{AB_{i+1}} = q_i \times M + S_{AB_{i+1}} \quad (15)$$

$$S_{qm} = \left(\sum_{i=0}^{k_1-1} q_i \times (d_A)^i \right) \times M + \left(\sum_{i=0}^{k_1-1} b_i \times (d_A)^i \right) \times A = q \times M + B \times A \quad (16)$$

This paper introduces a method for the cross execution of modular multiplication and modular reduction operations using polynomial expansion. Unlike the classical Montgomery modular multiplication algorithm, the proposed algorithm performs modular multiplication with a smaller bit-width before initiating the modular reduction operation, thereby reducing the start time for modular reduction. These enhancements make the modular multiplication algorithm more suitable for hardware implementation involving large integer modular multiplication. By alternating the execution of modular multiplication and modular reduction using smaller bit-widths, computational efficiency is increased. The

new algorithm conserves hardware resources by minimizing the multiplier area, resulting in efficient modular multiplication with low resource consumption.

4.2. Parallel Montgomery Ladder Algorithm Based on Co-Z Operations

The Montgomery ladder algorithm based on Co-Z operations improves the efficiency of scalar multiplication by eliminating the need for Z coordinate calculations in the main loop. In the Montgomery point multiplication algorithm, irrespective of the value of k_i , each iteration of the loop involves computing both point addition and point doubling, which are independent operations that can be executed in parallel. To optimize these operations, Co-Z operations are introduced. When the Z coordinates of two different points are equal in Jacobian projective coordinates, the addition operation can omit certain calculations involving the Z coordinate, thus enhancing the efficiency of the addition operation. The ladder step in the main loop of the traditional Montgomery ladder algorithm is replaced with the following.

First, perform one Co-Z conjugate addition operation, then perform one Co-Z addition operation. The algorithm proceeds as follows:

1. $a = (k_i + k_{i+1}) \bmod 2$
2. $\text{XYCZ_addC}(R_{1-a}, R_a);$
3. $\text{XYCZ_add}(R_a, R_{1-a});$

This method restores the Z coordinate at the final stage of the Montgomery ladder algorithm with minimal additional cost. The optimized point multiplication operation reduces the time complexity by the equivalent of 6 modular multiplications.

To fully leverage the parallelism of the SIMD instruction set, we carefully analyzed the execution flow of Co-Z Jacobian operations. We found that most prime field operations lack data dependencies and can be executed in parallel. Algorithms 4 and 5 provide parallel optimized implementations of Co-Z addition and Co-Z conjugate addition, respectively, involving only the (X, Y) coordinates. These algorithms implement elliptic curve point operations using parallel prime field operations with AVX2. In these algorithms, each line represents the simultaneous execution of the same operation on two sets of data. However, during the parallelization of Co-Z operations, some operations in Algorithm 4 cannot be paired; these are relatively simple addition and subtraction operations. Conversely, Algorithm 5 ensures that all operations can be paired by precomputing A' and T' .

Algorithm 4 PCZ-ADD: Parallel Co-Z Addition Operation

Input: $P = (X_P, Y_P), Q = (X_Q, Y_Q)$

Output: $(R, R') = (P + Q, P')$

#	Operation	Comment
1	$T_1 = X_P - X_Q$ $T_2 = Y_Q - Y_P$	{mod sub}
2	$A = T_1^2$ $D = T_2^2$	{mod square}
3	$B = X_P A$ $C = X_Q A$	{mod mul}
4	$T_1 = B + C$	{mod add}
5	$X_R = D - T_1$ $T_3 = C - B$	{mod sub}
6	$T_1 = B - X_R$	{mod sub}
7	$T_1 = T_1 T_2$ $E = Y_P T_3$	{mod mul}
8	$Y_R = T_1 - E$	{mod sub}
9	return $((X_R, Y_R), (B, E))$	

Compared with the sequential implementation of Co-Z addition, the time complexity of parallel Co-Z addition is nearly halved, reducing from $4\dot{M} + 2\dot{S} + 7\dot{A}$ to $2\ddot{M} + 1\dot{S} + 5\ddot{A}$, where \ddot{M} , \dot{S} , and \ddot{A} represent bidirectional parallel modular multiplication, modular squaring, and modular addition/subtraction operations, respectively. The counts of modular multiplication and modular squaring are thus reduced by 50%. Additionally, the bidirec-

tional parallel scheduling of the Co-Z conjugate addition lowers the time complexity from $5M + 3S + 11A$ to $3\ddot{M} + 1\ddot{S} + 6\ddot{A}$.

Algorithm 5 PCZ-ADDC: Parallel Co-Z Conjugate Addition Operation

Input: $P = (X_P, Y_P)$, $Q = (X_Q, Y_Q)$, $A' = (X_Q - X_P)^2$, $T' = (X_Q - X_P)A' = C' - B'$

Output: $(R, R') = (P + Q, P - Q)$

#	Operation	Comment
1	$C = X_Q A'$ $E = Y_P T'$	{mod mul}
2	$B = C - T'$ $T_1 = Y_Q - Y_P$	{mod sub}
3	$T_2 = B + C$ $T_3 = Y_P + Y_Q$	{mod add}
4	$D = T_1^2$ $F = T_3^2$	{mod square}
5	$X_R = D - T_2$ $X'_R = F - T_2$	{mod sub}
6	$T_2 = B - X_R$ $T_4 = X'_R - B$	{mod sub}
7	$T_2 = T_2 T_1$ $T_3 = T_3 T_4$	{mod mul}
8	$Y_R = T_2 - E$ $Y'_R = T_3 - E$	{mod sub}
9	return $((X_R, Y_R), (X'_R, Y'_R))$	

Analyzing the computational processes in Co-Z addition and Co-Z conjugate addition reveals a total time complexity of $9M + 5S + 18A$. Rivain [28] proposed an optimization scheme that replaces one multiplication with one squaring and four additions, reducing the Co-Z ladder algorithm's time complexity to $8M + 6S + 22A$. Since these operations are even-numbered in prime fields, theoretically, the lowest time complexity for a bidirectional parallel Co-Z ladder algorithm is $4\ddot{M} + 3\ddot{S} + 11\ddot{A}$. However, the total time complexity for sequential execution of Algorithms 4 and 5 is $5\ddot{M} + 2\ddot{S} + 11\ddot{A}$, which does not reach the theoretical minimum. Therefore, we merged Co-Z conjugate addition and Co-Z addition to achieve the lowest theoretical time complexity. Before entering the main loop, we precompute two large integers, $A' = (X_Q - X_P)^2$ and $T' = (X_Q - X_P)A'$, and repeatedly update them during the main loop. This significantly reorders the operations in the Co-Z ladder algorithm to reduce the parallelized algorithm's time complexity to $4\ddot{M} + 3\ddot{S} + 13\ddot{A}$, very close to the theoretical minimum, requiring only two additional modular addition operations. Detailed steps are in Algorithm 6. Except for the last two operations, all can be processed in parallel, making Algorithm 6 highly suitable for bidirectional parallel implementation. Compared with the sequential combined implementation of Co-Z conjugate addition and Co-Z addition with a time complexity of $5\ddot{M} + 2\ddot{S} + 11\ddot{A}$, our optimization reduces the Co-Z ladder algorithm's time complexity to the minimum. Our scheme replaces $1\ddot{M}$ with $1\ddot{S} + 2\ddot{A}$. By comparing the clock cycles of bidirectional parallel modular multiplication and squaring operations, our scheme effectively enhances the efficiency of the Co-Z ladder algorithm. The overall time complexity optimization effect of the improved algorithm is compared, as shown in Table 1.

Table 1. Improved time complexity of the parallel Co-Z algorithm.

Algorithm	Sequential Execution	Parallel Execution
Co-Z Addition	$4M + 2S + 7A$	$2\ddot{M} + 1\ddot{S} + 5\ddot{A}$
Co-Z Combined Addition	$5M + 3S + 11A$	$3\ddot{M} + 1\ddot{S} + 6\ddot{A}$
Sequential Execution	$9M + 5S + 18A$	$5\ddot{M} + 2\ddot{S} + 11\ddot{A}$
Precomputation A', T'	$8M + 6S + 22A$	$4\ddot{M} + 3\ddot{S} + 13\ddot{A}$

Algorithm 7 demonstrates the complete process of the Montgomery ladder algorithm based on parallel Co-Z operations. This algorithm initializes two elliptic curve points $(R_1, R_0) = (2P, P)$ using the INIT algorithm, making their Z coordinates equal, allowing them to use Co-Z operations.

Algorithm 6 PCZ-LADDER: Parallel Co-Z Ladder Algorithm

Input: $P = (X_P, Y_P) = R_a, Q = (X_Q, Y_Q) = R_{1-a}, A' = (X_Q - X_P)^2, T' = (X_Q - X_P)A' = C' - B', a \in \{0, 1\}, R_a, R_{1-a}$ are two Co-Z coordinates points updated in the Montgomery ladder algorithm.

Output: $(R_a, R_{1-a}) = (2R_a, R_a + R_{1-a})$ with updated $A' = (X_{R_a} - X_{R_{1-a}})^2$ and $T' = (X_{R_{1-a}} - X_{R_a})A'$

#	Operation	Comment
1	$C' = X_Q A' \quad E' = Y_P T'$	{mod mul}
2	$B' = C' - T' \quad T_1 = Y_Q - Y_P$	{mod sub}
3	$T_2 = B' + C' \quad T_3 = Y_P + Y_Q$	{mod add}
4	$D' = T_2^2 \quad F' = T_3^2$	{mod square}
5	$X_R = D' - T_2 \quad X'_R = F' - T_2$	{mod sub}
6	$T_2 = B' - X_R \quad T_4 = X'_R - B'$	{mod sub}
7	$T_2 = T_2 T_1 \quad T_4 = T_3 T_4$	{mod mul}
8	$Y_R = T_2 - E' \quad Y'_R = T_4 - E'$	{mod sub}
9	$T_1 = X'_R - X_R \quad T_2 = Y'_R - Y_R$	{mod sub}
10	$A = T_1^2 \quad D = T_2^2$	{mod square}
11	$X_P = B = X_R A \quad C = X'_R A$	{mod mul}
12	$T_3 = T_2 + B \quad T_4 = B + C$	{mod add}
13	$X_Q = D - T_4 \quad T_1 = C - B$	{mod sub}
14	$T_4 = X_Q - X_P \quad T_3 = T_3 - X_Q$	{mod sub}
15	$A' = T_4^2 \quad T_3 = T_3^2$	{mod square}
16	$T' = T_4 A' \quad X_P = E = Y_R T_1$	{mod mul}
17	$T_1 = D + A' \quad T_2 = E + E$	{mod add}
18	$T_3 = T_3 - T_1$	{mod sub}
19	$Y_Q = \frac{1}{2}(T_3 - T_2)$	{mod sub}
20	return $((X_Q, Y_Q), (X_P, Y_P))$	

Algorithm 7 Montgomery Ladder Algorithm Based on Co-Z Operation

Input: Elliptic curve point $P \neq \infty$, a scalar $k \in \mathbb{F}_p$, and $k_{n-1} = 1$.

Output: The result of the scalar multiplication $k \cdot P$.

```

1:  $(R_1, R_0) = \text{INIT}(P)$ 
2:  $A' = (X_0 - X_1)^2, T' = (X_0 - X_1)A'$ 
3: for  $i$  from  $n - 2$  by 1 down to 0 do
4:    $a = (k_i + k_{i+1}) \bmod 2$ 
5:    $(R_a, R_{1-a}, A', T') = \text{PCZ-LADDER}(R_a, R_{1-a}, A', T')$ 
6: end for
7:  $a = (k_0 + k_1) \bmod 2$ 
8:  $(R_a, R_{1-a}) = \text{PCZ-ADDC}(R_a, R_{1-a}, A', T')$ 
9:  $\frac{\lambda}{Z} = \text{Final\_Inv\_Z}(R_{1-a}, R_a, P, a)$ 
10:  $(R_0, R_1) = \text{PCZ-ADD}(R_0, R_1)$ 
11: return  $((\frac{\lambda}{Z})^2 X_0, (\frac{\lambda}{Z})^3 Y_0)$ 

```

After initializing the elliptic curve points R_1 and R_0 , we precompute the items $A' = (X_Q - X_P)^2$ and $T' = (X_Q - X_P)A'$ mentioned in the Co-Z ladder algorithm to adjust the calculation order in the main loop and reduce the time complexity. Entering the main loop of the Montgomery ladder algorithm, we replace the traditional trapezoidal algorithm $P_1 = 2P_1; P_2 = P_1 + P_2$ with the following equations:

$$(R_{1-a}, R_a) = (R_a + R_{1-a}, R_a - R_{1-a})$$

$$R_a = R_{1-a} + R_a$$

where $a = ([k_i + k_{i+1}] \bmod 2)$. As shown in Algorithm 7, the Montgomery ladder algorithm based on parallel Co-Z operations executes one parallel Co-Z ladder algorithm per iteration, thus having a regular execution flow and fixed execution time. The precomputed items A' and T' are updated once in each iteration of the Co-Z ladder algorithm. After the main loop ends, we need to convert the elliptic curve points from Co-Z Jacobian projective

coordinates back to affine coordinates. This process is achieved through the Final_Inv_Z algorithm, which computes $Z = X_P Y_{R_0} (X_{R_0} - X_{R_1})$ and $\lambda = y_P X_{R_0}$, and outputs $\frac{\lambda}{Z}$. The algorithm's time complexity is $1I + 3M + 1A$, requiring one inversion operation in the prime field.

5. Evaluation

This section presents performance data of our optimized implementation of the SM2 cryptographic algorithm. We evaluate execution time and throughput as key performance metrics, and compare our approach with other available schemes (e.g., the SM2 signature algorithm and ECDSA (secp256k1) based on OpenSSL 1.1.1). We selected OpenSSL 1.1.1 (LTS) as it is a widely used version that supports server security in many industrial applications, making our evaluation more representative. The test environment is shown below, as presented in Table 2.

Table 2. Comparison of experimental hardware configurations.

Processor	Frequency	Memory	Operating System	SIMD Support
quad-core ARM Cortex-A7	900 MHz	1 GB	Raspbian OS	NEON
4th gen Intel Core i5-4402EC	2.5 GHz	8 GB	Ubuntu 18.04 LTS	AVX2

We conducted tests on both processors using the standard SM2 algorithm, the optimized SM2 algorithm, and ECDSA for encryption, decryption, signing, and verification operations. Each operation was performed 3000 times, and the average time was recorded. The time statistics do not include hashing time, and the data length for encrypted messages was set to 1 KB. The experimental results are shown in Table 3.

Table 3. Performance comparison between SM2 and our SM2 on different processors.

Operation	Processor	SM2 (ms)	Our SM2 (ms)	Improvement (%)
Signing	ARM Cortex-A7	2045.8	1634.3	20.1%
	Intel Core i5	1010.3	701.2	30.6%
Verification	ARM Cortex-A7	2189.2	1780.6	18.7%
	Intel Core i5	1078.3	850.1	21.2%
Encryption	ARM Cortex-A7	4285.3	3352.4	21.8%
	Intel Core i5	1997.5	1348.1	32.5%
Decryption	ARM Cortex-A7	2324.7	1867.2	19.7%
	Intel Core i5	1140.8	854.2	25.1%

As shown in the data, the optimization scheme brings significant performance improvements on both Intel Core i5 and low-power ARM Cortex-A7 processors. The improvement is especially notable on the Intel Core i5 processor, which supports more advanced SIMD instruction sets (AVX2). On the ARM processor, despite limited SIMD support, the optimization still achieves approximately 20% performance gains in resource-constrained environments.

For the comparison with other schemes, we primarily used the Intel Core i5 processor for testing. This choice is based on the following considerations:

- Intel Core i5 processor supports a wider range of SIMD instructions, which better demonstrate the performance advantage of the optimization scheme.
- The computational capability of this processor is commonly found in edge devices for IoT, providing a certain level of representativeness.

Therefore, in the following comparative analysis, we mainly base our discussion on the test results of the Intel processor.

Utilizing precomputation techniques is an effective way to enhance the performance of SM2 algorithm implementation. Precomputation significantly improves program efficiency. Many protocols in the SM2 public key cryptosystem require repeated point multiplications of the base point “ G ”. In our implementation, we precompute these multiples of the base point G and store them in the code, allowing for quick lookup and retrieval during runtime, as summarized in Table 4.

Table 4. SM2 base point calculation table.

i	j	Storage Point	i	j	Storage Point
0	1	1G	2	1	$2^{16}G$
0	2	2G	2	2	$2^{16} \times 2G$
...
0	255	$255G$	2	255	$2^{16} \times 255G$
1	1	2^8G
1	2	$2^8 \times 2G$	31	1	$2^{31}G$
...
1	255	$2^8 \times 255G$	31	255	$2^{31} \times 255G$

In the SM2 signature algorithm, it is necessary to calculate $s = ((1 + d_A)^{-1} \cdot (k - r \cdot d_A)) \bmod n$, where $(1 + d_A)^{-1}$ requires a modular inverse operation. To enhance efficiency when signing large numbers of data with a given key, the value of $(1 + d_A)^{-1}$ can be precomputed and subsequently utilized in each signing process.

Table 5 compares the number of operations required for point addition and point doubling in various coordinate systems. Here, I, M, and S denote modular inversion, modular multiplication, and modular squaring, respectively. It is evident that Jacobian coordinates offer superior performance compared with other methods. For point addition, the computation requires 12 modular multiplications and four modular squarings; similarly, for point doubling, the computation also requires 12 modular multiplications and four modular squarings.

Table 5. Comparison of point addition and doubling operations in different coordinate systems.

	Affine Coordinates	Standard Projective Coordinates	Jacobian Coordinates
Point Addition	$I + 2M + S$	$13M + 2S$	$12M + 4S$
Doubling	$I + 2M + 2S$	$8M + 5S$	$4M + 6S$

I: Modular inversion. Computationally expensive operation, especially in large field arithmetic.

M: Modular multiplication. Requires significant computational resources but fewer than modular inversion.

S: Modular squaring. Special case of modular multiplication, generally faster but still computationally significant.

Compared with the commonly used affine coordinates and standard projective coordinates, Jacobian coordinates eliminate the need for modular inversion operations. In affine coordinates, point addition requires one modular inversion, which is typically computationally intensive. However, Jacobian coordinates enable point addition and point doubling without any inversion operations, significantly improving computational efficiency. Moreover, Jacobian coordinates exhibit higher efficiency in point doubling operations due to fewer required multiplications and squarings. While standard projective coordinates also avoid modular inversion and allow for simpler mathematical representation, they generally incur higher arithmetic overhead. Additionally, Jacobian coordinates are highly compatible with Montgomery point multiplication, further enhancing computational efficiency.

5.1. Montgomery Modular Multiplication

For Montgomery modular multiplication, we optimized the classical Montgomery modular multiplication algorithm to better suit hardware implementations for large integer modular multiplication. This improved algorithm alternates between modular multiplication and modular reduction with a small bit-width, thereby enhancing computational efficiency. By reducing the multiplier's size, the new algorithm minimizes hardware resource usage, achieving efficient modular multiplication with low resource consumption. We performed comparative evaluations between the fast modular multiplication scheme based on the improved Montgomery algorithm and several other public algorithms. Given that modular multiplication accounts for approximately 70% of the computational overhead in the entire SM2 algorithm, we tested the runtime of the entire SM2 protocol using randomly generated public–private key pairs. The results are presented in Table 6.

Table 6. Performance test results of SM2 algorithm under different modular multiplication algorithms.

Modular Multiplication Algorithm	SM2 Signature Time (s)	SM2 Verification Time (s)	SM2 Encryption Time (s)	SM2 Decryption Time (s)
Montgomery Mult [10]	0.6528	1.2938	1.2763	0.6490
SOS [11]	0.2581	0.4988	0.5039	0.2182
FIOS [11]	0.3696	0.7224	0.7221	0.3671
Literature [29]	0.4129	0.8105	0.8070	0.4103
Literature [30]	0.2199	0.4220	0.4292	0.2182
This Work	0.2065	0.3951	0.4024	0.2045

5.2. Parallel Montgomery Ladder Algorithm

Table 7 presents the execution times for the sequential and parallel implementations of Co-Z addition, Co-Z conjugate addition, Co-Z ladder algorithm, and the Montgomery ladder algorithm based on Co-Z operations. Our parallel implementations significantly outperform the corresponding sequential implementations, with speedup ratios ranging from 1.26 to 1.60. Specifically, the bidirectional parallel implementation of the Montgomery ladder algorithm based on Co-Z operations achieves an execution time of 274,908 clock cycles, making it 1.31 times faster than its sequential counterpart.

Table 7. Comparison of clock cycles for sequential and parallel implementation of Co-Z operations and Montgomery ladder based on Co-Z operations.

Implementation Method	Co-Z Addition	Co-Z Diagonal Addition	Co-Z Ladder Algorithm	Montgomery Ladder Algorithm
Sequential Implementation	555	786	1,334	359,868
Parallel Implementation	439	489	1,001	274,908
Speedup Ratio	1.26	1.60	1.33	1.31

Table 8 compares the time complexity and clock cycles of the Montgomery ladder algorithm based on Co-Z operations on the SM2, Curve25519, and NIST P-256 curves. Since SM2 and Curve25519 utilize Co-Z coordinates, modular inversion is not required in scalar multiplication, significantly reducing computational complexity. Curve25519, as an efficient Montgomery curve, further optimizes performance by supporting operations with only the X coordinate. Additionally, the mathematical properties and precomputation requirements of each curve influence performance. SM2, based on the short Weierstrass form, requires additional coefficient calculations and affine transformations, while Curve25519, adopting the Montgomery form, simplifies computational complexity. Overall, Curve25519

demonstrates the best performance, while SM2 on this processor also performs within the expected range; in contrast, NIST P-256, without Co-Z optimization, shows higher clock cycle requirements. (In this context, “C” represents a single multiplication by a curve constant, which is typically much faster than a standard modular multiplication.)

Table 8. Comparison of time complexity and clock cycles for AVX2 implementation of Montgomery ladder based on Co-Z operations and other fixed-point multiplication algorithms.

Implementation Method	Time Complexity per Bit	Additional Time Complexity	Clock Cycles
SM2 (This Work)	$4\ddot{M} + 3\ddot{S} + 13\ddot{A}$	$I + 8M + 7S + 12A$	274,908 (CL)
Curve25519	$3\ddot{M} + 2\ddot{S} + 1\ddot{C} + 4\ddot{A}$	$I + 1M$	156,500 (H)
NIST P-256	n/a	n/a	312,000 (H)

5.3. Performance of Digital Signature Algorithm Implementations

In this section, we conduct comparative testing on the SM2 signature algorithm based on OpenSSL 1.1.1, ECDSA (secp256k1), work [30], work [31], and our implemented SM2 signature algorithm to evaluate the overall performance of the optimized SM2 cryptographic algorithm designed in this study. The time consumed by the hashing process in these signature algorithms is excluded from the time statistics. To minimize statistical errors, we performed 10 rounds of tests for each algorithm, with each round comprising 3000 signatures and verifications. In the final statistics, the maximum and minimum values from each round were discarded, and the average was calculated to obtain the final test results for 3000 signatures and verifications, as shown in Table 9.

In signature and verification operations, the time consumed by the hashing algorithm increases as the data size grows. As the number of data to be tested increases, the proportion of time spent on the hashing process in the overall signature and verification operations also gradually increases. Consequently, in cases with large data sizes, the total time for signature and verification is dominated by the hashing time. To eliminate the impact of SM3 and SHA256 on the test duration, we excluded the hashing process in this section’s tests. The essence of a digital signature is to use a public-key cryptographic scheme to encrypt the hash value to obtain a data digest. Therefore, by excluding the hashing process, the tests effectively measure the encryption of the data digest directly. Furthermore, since both SM3 and SHA256 produce a fixed output length of 256 bits, the length of the data digest remains constant, indicating that the test results in this section are independent of the length of the data to be signed.

Table 9. Time consumption for 3000 signatures without hashing.

Time Consumption (ms)	SM2	Our SM2	ECDSA	Work [30]	Work [31]
Signature	1012.0	705.7	1087.2	802.5	827.9
Verification	1079.2	870.4	1207.3	902.3	944.8

The data from Table 9 allow us to derive a performance comparison chart for signature and verification, depicted in Figure 1. The horizontal axis indicates the time consumed for 3000 signatures or verifications by each algorithm, measured in milliseconds.

The data in Table 9 and Figure 1 show that our optimized SM2 implementation demonstrates superior performance in both signing and verification operations compared with other algorithms and implementations. For 3000 signature operations, our SM2 implementation requires 705.7 milliseconds, significantly less than the 1012.0 milliseconds for the standard SM2 and 1087.2 milliseconds for ECDSA, highlighting its higher signing efficiency. Similarly, for verification, our SM2 implementation achieves a time of 870.4 milliseconds, which is faster than both the standard SM2 at 1079.2 milliseconds and ECDSA at 1207.3 milliseconds. Compared with other works (work [30] and work [31]), our implementation consistently shows

lower time consumption for both signing and verification. This improvement is particularly noticeable because the SM2 signing algorithm involves more point doubling operations on the base point “G” than the verification algorithm. By precomputing the base point “G”, we achieve a more significant speedup in this aspect, resulting in a noticeable difference in the time required for signing and verification.

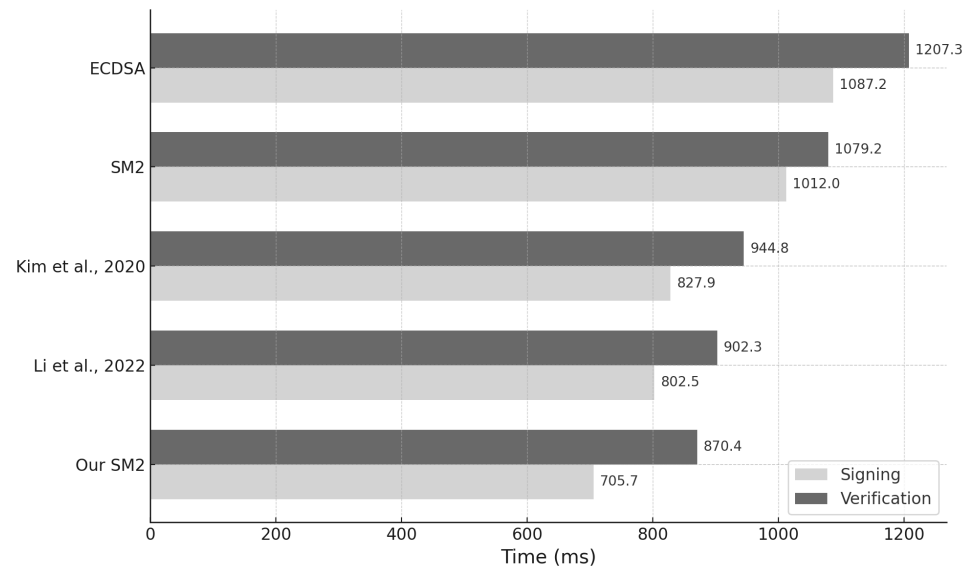


Figure 1. Performance comparison of SM2 digital signature implementations.

5.4. Performance of SM2 Data Encryption Algorithm Implementation

This section presents performance overhead tests for our implemented SM2 data encryption algorithm. We tested encryption on data sizes of 32 B, 64 B, 128 B, 256 B, 512 B, 1 KB, 2 KB, 4 KB, 8 KB, 16 KB, 32 KB, 64 KB, 128 KB, and 256 KB. For each data size, randomly generated data were used to eliminate potential biases.

The performance testing followed these steps: (1) randomly generate the test data; (2) encrypt and decrypt each group of test data 300 times, repeating this process 10 times; and (3) exclude the maximum and minimum values from each group’s results and calculate the average to obtain the final test result for that data size.

For comparative purposes, we selected ECIES (secp256k1) as a reference algorithm in addition to our implemented SM2 data encryption algorithm. The SM3 algorithm was used as the hash function within the SM2 algorithm. The organized test results are presented in Tables 10 and 11.

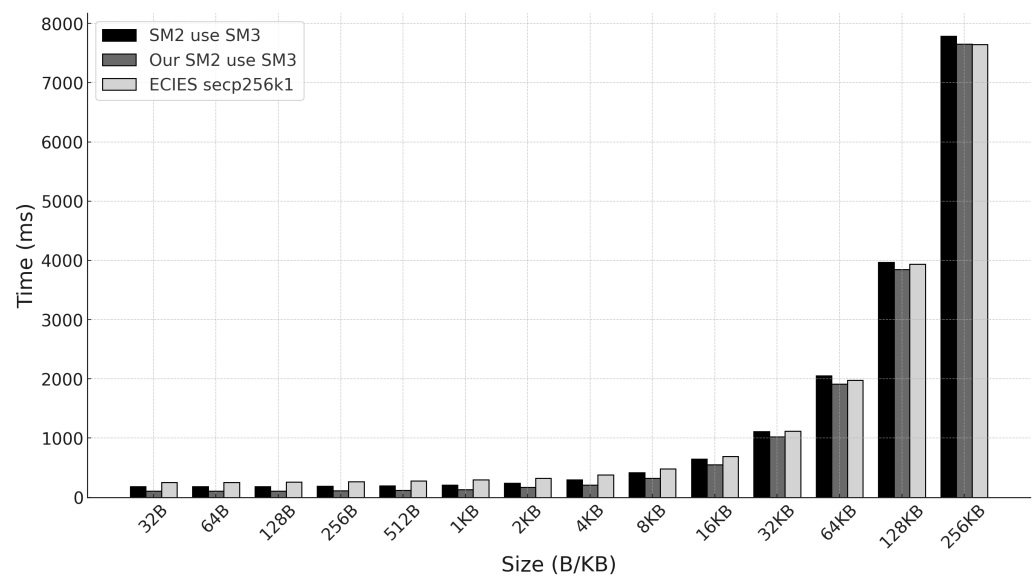
Table 10. The duration of 300 encryption processes.

Time/ms	SM2 use SM3	Our SM2 Use SM3	ECIES secp256k1
32 B	181.0	104.3	249.1
64 B	181.6	104.5	254.4
128 B	183.6	103.9	260.0
256 B	187.3	109.1	267.4
512 B	194.8	116.7	278.6
1 KB	210.3	131.5	295.1
2 KB	239.5	166.8	324.2
4 KB	298.1	207.6	378.6
8 KB	415.4	320.1	483.4
16 KB	647.4	549.2	693.4
32 KB	1113.4	1021.2	1116.2
64 KB	2054.0	1912.6	1976.9
128 KB	3969.7	3847.3	3938.0
256 KB	7784.4	7654.8	7644.2

Table 11. The duration of 300 decryption processes.

Time/ms	SM2 Use SM3	Our SM2 Use SM3	ECIES secp256k1
32 B	97.4	73.2	116.2
64 B	98.3	73.9	124.3
128 B	99.5	74.8	130.8
256 B	100.7	75.7	138.7
512 B	104.9	78.8	145.2
1 KB	113.5	85.3	159.3
2 KB	129.2	97.1	185.2
4 KB	159.8	120.0	206.6
8 KB	222.9	167.3	308.5
16 KB	347.4	260.7	387.2
32 KB	556.6	447.6	541.0
64 KB	994.4	821.0	849.5
128 KB	1704.3	1578.4	1528.0
256 KB	3546.9	3165.3	3061.6

From Figures 2 and 3, it is clear that our implemented SM2 data encryption algorithm improves encryption and decryption speeds compared with the original implementation. The time overhead for the signing and verification process is reduced by approximately 20% compared with the existing OpenSSL implementation, while the time overhead for data encryption and decryption is reduced by about 25% for smaller data sizes. The proportion of overhead from the hash function in the data encryption algorithm increases with the size of the data to be encrypted, and the time consumption of the encryption algorithms using the same hash function tends to converge as data size increases.

**Figure 2.** The duration of 300 encryption processes.

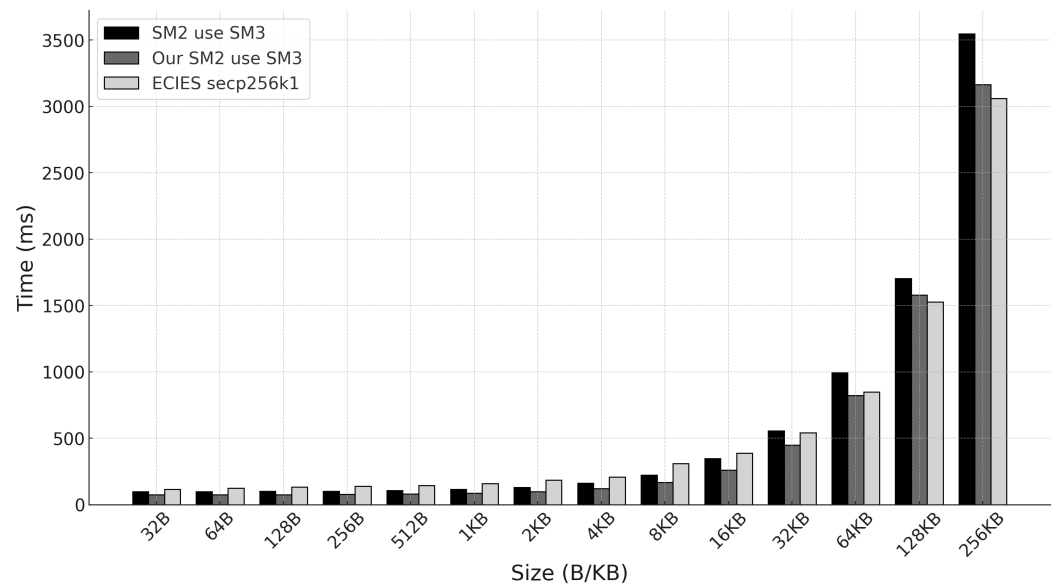


Figure 3. The duration of 300 decryption processes.

6. Discussion

Although our work has achieved significant improvements in the computational efficiency of the SM2 algorithm, several limitations remain.

First, while the proposed optimizations effectively reduce execution time, additional factors, such as power consumption and memory requirements, also influence the adoption of the algorithm in real-world IoT devices. Experimental results show that memory usage remains largely unchanged, as the reduction in computational redundancy is counter-balanced by the increased demand from SIMD parallel instruction sets. However, the parallelism achieved through SIMD instructions may lead to higher power consumption during cryptographic operations, which is a critical concern for resource-constrained IoT environments. Balancing execution speed, power consumption, and memory usage will be a focus of future research. For example, exploring low-power architectures and optimizing instruction-level parallelism could mitigate the increased energy demands, while adaptive strategies that adjust computational methods dynamically based on device contexts may further enhance practical applicability.

Second, although resilience to side-channel attacks is a critical consideration for cryptographic algorithms, especially in IoT devices, this study primarily focuses on optimizing computational efficiency through software-level improvements. Techniques such as constant-time algorithms, scalar randomization, and secure memory access are commonly used to address side-channel vulnerabilities [31–34]. However, integrating such countermeasures typically involves trade-offs between security and performance, which were beyond the scope of this work. Future studies will explore software-based strategies to enhance side-channel attack resistance, such as algorithmic masking and timing attack mitigations, aiming to improve the security of the optimized SM2 algorithm without significantly compromising its performance gains.

In the future, we aim to investigate solutions that simultaneously address the above limitations. By integrating power-efficient optimizations and side-channel countermeasures, the proposed algorithm can be further refined to meet the stringent requirements of IoT applications, ensuring both high performance and robust security.

7. Conclusions

With the ongoing increase in computational power, traditional ECC algorithms are encountering performance bottlenecks. To address these issues, we optimized elliptic curve group computation by introducing a cross Montgomery modular multiplication algorithm based on polynomial expansion and leveraging SIMD technology to accelerate

the Montgomery ladder algorithm using Co-Z operations. This approach enhances ECC performance and bolsters information security. By segmenting large integer operations into polynomials, we perform modular multiplication and reduction simultaneously, reducing storage demands and streamlining the Montgomery multiplication process. The incorporation of SIMD technology enables parallel processing, significantly boosting computational efficiency through optimized instruction scheduling and loop unrolling, which minimizes control overhead and enhances micro-level execution. Our results show that the optimized SM2 digital signature algorithm reduces computation time by approximately 20%, while the SM2 data encryption algorithm achieves a 25% performance improvement in encryption and decryption times for smaller data sizes. These optimizations advance ECC performance and provide strong support for information security.

Author Contributions: Conceptualization, H.Z. and Y.S. (Yubo Song); methodology, H.Z. and Y.S. (Yubo Song); software, H.Z. and D.L.; validation, H.Z. and Y.S. (Yizhen Sun); investigation, Y.S. (Yizhen Sun); formal analysis, H.Z.; data curation, Y.S. (Yizhen Sun); resources, H.Z.; writing—original draft, H.Z. and D.L.; writing—review and editing, Q.C.; visualization, D.L. and Q.C.; supervision, H.Z. and Q.C.; project administration, H.Z. and Z.T.; funding acquisition, Z.T. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported by State Grid Electric Power Co., Ltd. research project No. 5700-202323607A-3-2-ZN and Hunan Key Laboratory for Internet of Things in Electricity, P.R.China No. 2019TP1016.

Data Availability Statement: The data used in this study were randomly generated fixed-size text files, which were used solely for testing the performance of the optimized SM2 algorithm. As these datasets do not have intrinsic research value, they have not been publicly archived.

Conflicts of Interest: Authors Hongyu Zhu, Yizhen Sun, and Zheng Tian were employed by the State Grid Hunan Electric Power Company Limited Information and Communication Company, Qian Chen was employed by State Grid Hunan Electric Power Co., Ltd. The remaining authors declare that the research was conducted in the absence.

References

1. Rivest, R.L.; Shamir, A.; Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* **1978**, *21*, 120–126. [CrossRef]
2. Fagoyinbo, J.B. *The Armed Forces: Instrument of Peace, Strength, Development and Prosperity*; Author House: Bloomington, IN, USA, 2013.
3. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. *Satoshi Nakamoto* 2008. Available online: <https://static.upbitcare.com/931b8bfc-f0e0-4588-be6e-b98a27991df1.pdf> (accessed on 18 November 2024).
4. Hisil, H.; Egrice, B.; Yassi, M. Fast 4 Way Vectorized Ladder for the Complete Set of Montgomery Curves. *IACR Cryptology ePrint Archive* **2020**, 2020/388. Available online: <https://eprint.iacr.org/2020/388.pdf> (accessed on 11 June 2024).
5. Perlroth, N. Government announces steps to restore confidence on encryption standards. *The New York Times*, 11 September 2013.
6. Barker, E.B.; Kelsey, J.M. *Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revised)*; US Department of Commerce, Technology Administration, National Institute of Standards and Technology, Computer Security Division, Information Technology Laboratory: Washington, DC, USA, 2007.
7. Li, M.; Li, N.; Liu, H.; Cheng, S.; Hu, X.; Li, J. Implementation of a Safe and Efficient Point Multiplication for SM2 Algorithm. In Proceedings of the 2023 IEEE 6th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), Chongqing, China, 24–26 February 2023; pp. 137–141. [CrossRef]
8. Liu, D.; Yuan, Y.; Xue, T.; Chen, L. An Efficient Batch Verification Scheme for SM2 Signatures. In Proceedings of the 2021 8th International Conference on Dependable Systems and Their Applications (DSA), Yinchuan, China, 5–6 August 2021; pp. 208–212. [CrossRef]
9. Chinbat, M.; Wu, L.; Batsukh, A.; Khuchit, U.; Zhang, X.; Mongolyn, B.; Xu, K.; Yang, W. Performance Comparison of Finite Field Multipliers for SM2 Algorithm based on FPGA Implementation. In Proceedings of the 2020 IEEE 14th International Conference on Anti-counterfeiting, Security, and Identification (ASID), Xiamen, China, 30 October–1 November 2020; pp. 69–72. [CrossRef]
10. Montgomery, P.L. Modular multiplication without trial division. *Math. Comput.* **1985**, *44*, 519–521. [CrossRef]
11. Koc, C.; Acar, T.; Bailey, R., Jr. Analyzing and comparing Montgomery multiplication algorithms. *IEEE Micro* **1996**, *16*, 26–33. [CrossRef]
12. Montgomery, P.L. Speeding the Pollard and elliptic curve methods of factorization. *Math. Comput.* **1987**, *48*, 243–264. [CrossRef]

13. López, J.; Dahab, R. Fast multiplication on elliptic curves over GF(2m) without precomputation. In Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems, Berlin, Germany, 12–15 August 1999; Springer: Berlin/Heidelberg, Germany, 1999; pp. 316–327.
14. Longa, P.; Gebotys, C. Efficient techniques for high-speed elliptic curve cryptography. In Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems, Berlin, Germany, 17–20 August 2010; Springer: Berlin/Heidelberg, Germany, 2010; pp. 80–94.
15. Zhang, X.C.; Guo, H.; Zhang, X.Y.; Wang, C.; Liu, J. Fast Software Implementation of SM4. *J. Cryptologic Res.* **2020**, *7*, 799–811. [\[CrossRef\]](#)
16. Zhang, Y.P.; Gao, Y.; Yan, Y.; Liu, X. Fast Software Implementation of ZUC Algorithm. *J. Cryptologic Res.* **2021**, *8*, 388–401. .
17. Nath, K.; Sarkar, P. Efficient 4-way vectorizations of the Montgomery ladder. *IEEE Trans. Comput.* **2022**, *71*, 712–723. [\[CrossRef\]](#)
18. Cheng, H.; Großschädl, J.; Tian, J.Q.; Rønne, P.B.; Ryan, P.Y. High-Throughput Elliptic Curve Cryptography Using AVX2 Vector Instructions. In *Selected Areas in Cryptography—SAC 2020*; Springer: Cham, Switzerland, 2021; pp. 698–719. [\[CrossRef\]](#)
19. Huang, J.H.; Liu, Z.; Hu, Z.; Großschädl, J. Parallel Implementation of SM2 Elliptic Curve Cryptography on Intel Processors with AVX2. In *Australasian Conference on Information Security and Privacy*; Springer: Cham, Switzerland, 2020; pp. 204–224. [\[CrossRef\]](#)
20. Gueron, S.; Krasnov, V. Fast Prime Field Elliptic-Curve Cryptography with 256-Bit Primes. *J. Cryptogr. Eng.* **2015**, *5*, 141–151. [\[CrossRef\]](#)
21. Bernstein, D.J.; Duif, N.; Lange, T.; Schwabe, P.; Yang, B.Y. High-Speed High-Security Signatures. *J. Cryptogr. Eng.* **2012**, *2*, 77–89. [\[CrossRef\]](#)
22. Faz-Hernández, A.; López, J.; Dahab, R. High-Performance Implementation of Elliptic Curve Cryptography Using Vector Instructions. *ACM Trans. Math. Softw.* **2019**, *45*, 1–35. [\[CrossRef\]](#)
23. Chen, F.; Liu, Y.; Zhang, T.; Xu, Y.; Huang, Z. SM2-based low-cost and efficient parallel modular multiplication. *Microprocess Microsyst.* **2022**, *94*, 104650. [\[CrossRef\]](#)
24. Wang, D.; Lin, Y.; Hu, J.; Li, J.; Wang, Z. FPGA Implementation for Elliptic Curve Cryptography Algorithm and Circuit with High Efficiency and Low Delay for IoT Applications. *Micromachines* **2023**, *14*, 1037. [\[CrossRef\]](#) [\[PubMed\]](#)
25. Li, X.; Yu, W.; Li, B. Parallel and Regular Algorithm of Elliptic Curve Scalar Multiplication over Binary Fields. *Secur. Commun. Netw.* **2020**, *2020*, 4087873. [\[CrossRef\]](#)
26. Mai, L.; Yan, Y.; Jia, S.; Liao, X.; Liu, X. Accelerating SM2 digital signature algorithm using modern processor features. In Proceedings of the International Conference on Information and Communications Security, Cham, Switzerland, 19–22 November 2019; Springer: Cham, Switzerland, 2019; pp. 430–446.
27. Hu, X.; Zheng, X.; Zhang, S.; Chen, S.; Shi, Y. A high-performance elliptic curve cryptographic processor of SM2 over GF(p). *Electronics* **2019**, *8*, 431. [\[CrossRef\]](#)
28. Rivain, M. Fast and Regular Algorithms for Scalar Multiplication over Elliptic Curves. *IACR Cryptol. ePrint Arch.* **2011**, 2011/338. Available online: <https://eprint.iacr.org/2011/338> (accessed on 11 June 2024).
29. Yan, M. Fast implementation and optimization of SM2 digital signature and verification algorithm based on GPU. Ph.D. Thesis, Shanghai Jiaotong University, Shanghai, China, 2020.
30. Li, B.; Zhou, Q.L.; Chen, X.J.; Wu, L.; Zhang, Y. Optimization of reconfigurable SM2 algorithm over prime field. *J. Commun.* **2022**, *43*, 30–41.
31. Kim, K.H.; Choe, J.; Kim, S.Y.; Kim, N.; Hong, S. Speeding up regular elliptic curve scalar multiplication without precomputation. *Adv. Math. Commun.* **2020**, *14*, 703 [\[CrossRef\]](#)
32. Das, D.; Nath, M.; Chatterjee, B.; Ghosh, S.; Sen, S. STELLAR: A generic EM side-channel attack protection through ground-up root-cause analysis. In Proceedings of the 2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), McLean, VA, USA, 5–10 May 2019; pp. 11–20.
33. Das, D.; Danial, J.; Golder, A.; Ghosh, S.; Sen, S. EM and power SCA-resilient AES-256 through > 350× current-domain signature attenuation and local lower metal routing. *IEEE J. Solid-State Circuits* **2020**, *56*, 136–150. [\[CrossRef\]](#)
34. Das, D.; Maity, S.; Nasir, S.B.; Ghosh, S.; Raychowdhury, A.; Sen, S. High efficiency power side-channel attack immunity using noise injection in attenuated signature domain. In Proceedings of the 2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), Mclean, VA, USA, 1–5 May 2017; pp. 62–67.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.