Faster Software Implementation of the SIKE Protocol Based on A New Data Representation

Jing Tian, Piaoyang Wang, Zhe Liu, Jun Lin, Zhongfeng Wang, and Johann Großschädl

Abstract—Due to the smaller size in public and secret keys over other candidates for post-quantum cryptography (PQC), the supersingular isogeny key encapsulation (SIKE) protocol has survived from the second round fierce competition hosted by the National Institute of Standards and Technology (NIST) in January 2019. Many efforts have been done by researchers to reduce the computation latency, which, however, is still far more than desired. In the SIKE implementation, the Montgomery representation has been mostly adopted in the finite field arithmetic computing as the corresponding reduction algorithm is considered the fastest method for implementing the modular reduction. In this paper, we propose a new data representation for the supersingular isogeny-based elliptic-curve cryptography (ECC), of which the SIKE is a subclass. The new representation can facilitate faster modular reduction implementation than the Montgomery reduction. Meanwhile, the other finite field arithmetic operations in the ECC can also benefit from the proposed data representation. We have implemented all the arithmetic operations in C language with constant execution time based on our proposed data representation and applied them to the newest SIKE software library. Targeting at the SIKEp751, we run our design and the optimized implementation on a 2.6GHz Intel Xeon E5-2690 processor. The experiment results show that for the parameters of SIKEp751, the proposed modular reduction algorithm is about 2.61x faster than the best Montgomery one and our scheme also performs significantly better for the other finite field operations. With these improvements, the overall software implementation for the SIKEp751 achieves about 1.65x speedup compared to the state-of-the-art implementation.

Index Terms—Supersingular isogeny Diffie-Hellman (SIDH) key exchange, elliptic curve cryptography (ECC), modular reduction, Montgomery representation, Barrett reduction, post-quantum cryptography (PQC).

1 Introduction

In recent years, much progress has been made in quantum computers. The commonly used public-key cryptographic algorithms like the Rivest–Shamir–Adleman (RSA) [1] and elliptic-curve cryptography (ECC) [2] could be easily solved by using the Shor's algorithm [3] with powerful quantum computers. These achievements have indeed accelerated the development of post-quantum cryptography (PQC). From 2017, the National Institute of Standards and Technology (NIST) [4] has hosted two rounds of competition aiming to develop the post-quantum standards. The supersingular isogeny key encapsulation (SIKE) protocol [5] has survived from these competitions, as one of the 26 candidates. This protocol has the advantage of very short public and secret key sizes, which are perfectly compatible with the conventional ECC protocols. The SIKE protocol is developed by packaging the supersingular isogeny Diffie-Hellman (SIDH) key exchange protocol using the key encapsulation mechanism [6], to enhance the defense against various side-channel attacks [7], [8], [9]. The SIDH was first proposed by Jao and De Feo in 2011 to resist the quantum attack in terms of the difficulty to find isogenies between supersingular elliptic curves [10]. Generally, large degree isogenies are set to meet the security requirement. Long latency is caused mainly due to the serial computing of these isogenies, and it forms the bottleneck in practical applications. Therefore, methods to accelerate the SIDH can directly be applied to speed up the SIKE protocol.

Many researchers have conducted optimizations for the SIDH/SIKE protocol based on software [11], [12], [13], [14], [15], [16], [17] or hardware [18], [19], [20], [21], [22], [23], [24] platform. In the beginning, the software implementation of SIDH was done by Jao using the GMP library in 2011 [11], which presents the earliest version. The latest version provided in [17] is commonly recognized as the fastest software implementation, which has constantly integrated the state-of-the-art supersingular isogeny cryptographic schemes. Additionally, the library [17] covers the implementations for SIKE on x64, ARM, and FPGA platforms, which also combine the optimized methods proposed in the open literature. Many improvements have been made to speed up the SIKE protocol and make it more practical. However, those implementations for this PQC candidate are still more than one order of magnitude slower than many popular alternatives. It is noted that almost all of the implementations are based on the Montgomery representation. The major reason is that the associated reduction algorithm [25] has been widely regarded as the most efficient method to implement the modular reduction. Moreover, the Montgomery representation is also friendly to the computation of other field arithmetics.

A smooth prime of supersingular isogeny elliptic curves usually has the form of $p = f \cdot a^{e_A} b^{e_B} \pm 1$, where a and b are small primes, e_A and e_B are positive integers, and f is a small cofactor to make p prime. Considering the special structure of primes of those curves, some other efforts have been done to improve the performance of the modular reduction. In [26], Karmakar *et al.* proposed an efficient modular reduction for the prime of the form $p = 2 \cdot 2^{e_A} 3^{e_B} - 1$, where e_A and e_B must be even. They represented the field elements in quadratic form based on the unconventional radix $R = 2^{e_A/2} 3^{e_B/2}$ and derived a formula to replace the modulo-p operation with two modulo-R operations. Note that the two modular reductions are completed by the Barrett reduction algorithm [27] and the Montgomery method is not available here. Despite that, the new modular reduction algorithm achieves better performance than the conventional Montgomery one. Later, the modular reduction algorithm based on the unconventional quadratic representation has also been studied in [28] and [23]. Since the Montgomery reduction in multi-precision [15] only costs one multiplication for the general cases, the superiority of the algorithm in [26] fades away.

Our contributions: In this paper, we further explore the data representation based on a new unconventional radix for the supersingular isogeny elliptic curves. We transform the conventional structure of primes and extend the orders from quadratic to any orders. Based on this new data representation, we propose a low-complexity modular reduction algorithm with the improved Barrett reduction algorithm. The new method shows a big potential to outperform the best Montgomery reduction algorithm. It is worth to mention that this new data representation is also applicable to other finite field arithmetic operations. Thereafter, we implement the proposed algorithms in software and apply them to the SIKE protocol. The entire implementation is designed with constant time to defend the timing attack.

The main contributions are summarized as follows:

1) A general data representation of field elements is proposed for the supersingular isogeny elliptic curves, which can facilitate faster computation for field arithmetic operations.
2) An efficient modular reduction algorithm is derived based on the proposed data representation with several novel ideas. This new reduction algorithm obtains significant complexity reduction and it shows substantial superiority over the fastest Montgomery reduction algorithm when adopting a higher-order representation.
3) The other field operations are also deduced based on the proposed data representation. The operations on the new form require fewer computations than conventional ones.
4) A new SIKE implementation is presented by converting the input parameters to the new data representation and using the proposed field arithmetic algorithms.

We benchmark these implementations on a 2.6GHz Intel Xeon E5-2690 processor and set the order of representation to 12. All the tests of these algorithms have been passed of the parameters of SIKEp751 (NIST security level 5). The proposed modular reduction algorithm achieves 2.61x speedup over the fastest Montgomery reduction algorithm and the other tested field arithmetic operations are also computed faster than the previous works. Moreover, the entire implementation for SIKEp751 obtains a 1.65x faster speed than the optimized implementation provided in [17].

Paper organization: The rest of this paper is organized as follows. Section 2 gives a brief review of the SIDH and SIKE protocols, and basic field arithmetic operations. The proposed data representation and the efficient modular reduction algorithm are presented in Section 3. The finite field arithmetics based on the new data representation are derived in Section 4. In Section 5, the experiment results and comparisons are provided. Section 6 concludes this paper.

2 Preliminaries

The SIKE protocol is developed based on the SIDH key-exchange protocol by applying a key encapsulation mechanism [6] to enhance its anti-attack capability. By breaking down the computations of this protocol, it can be easily found that the large-degree isogeny computations occupy a dominant position. Those computations over elliptic curves can be divided into the basic arithmetic operations over a quadratic extension field $\mathbb{F}_{p^2}$, where p is a prime with a form of $f \cdot a^{e_A} b^{e_B} \pm 1$. And those operations can be further subdivided as arithmetic operations over $\mathbb{F}_p$. Usually, operations in $\mathbb{F}_p$ are processed on the Montgomery field, which has been widely considered providing better efficiency over others.

2.1 The SIDH Protocol

The SIDH key-exchange protocol is designed for two parties (saying, Alice and Bob) securely communicating with each other based on a shared secret key over a public communication environment. This shared secret is the j-invariant of two isomorphic supersingular elliptic curves generated based on a public supersingular elliptic curve E. The main steps of this protocol are summarized in Alg. 1, where the operations in the left-hand column are computed by Alice and those in the right by Bob. The public supersingular elliptic curve is usually set as the Montgomery curve with the form of

$$E/\mathbb{F}_{p^2} : Dy^2 = x^3 + Cx^2 + x, \tag{1}$$

where $C, D \in \mathbb{F}_{p^2}$, $D(C^2-4) \neq 0$, and $p = f \cdot a^{e_A} b^{e_B} \pm 1$. The two pairs of independent public points $\{P_A, Q_A\}$ and $\{P_B, Q_B\}$ all are on $E/\mathbb{F}_{p^2}$, and satisfy $<P_A, Q_A> = E[a^{e_A}]$ and $<P_B, Q_B> = E[b^{e_B}]$, respectively.

Algorithm 1: The SIDH key-exchange protocol [10].

```
Input: Public parameters: $E$, $(P_A, Q_A)$, and $(P_B, Q_B)$.
    Alice                                   Bob
1: Generate both parties' public and secret keys with the corresponding public parameters
    $sk_A \xleftarrow{\$} \{0,1\}^{e_A}$                   $sk_B \xleftarrow{\$} \{0,1\}^{e_B}$
    $pk_A \longleftarrow isogen_A(sk_A)$                 $pk_B \longleftarrow isogen_B(sk_B)$
2: Exchange the public keys and compute the shared secret key
    $j \longleftarrow isoex_A(pk_B, sk_A)$               $j \longleftarrow isoex_B(pk_A, sk_B)$
    $ss \longleftarrow H(j, M)$                          $ss \longleftarrow H(j, M)$
3: Communicate with each other by using the shared secret key
    $m_A \in \{0,1\}^M$                                  $m_B \in \{0,1\}^M$
    $c_A \longleftarrow ss \oplus m_A$                   $c_B \longleftarrow ss \oplus m_B$
    $m_B \longleftarrow c_B \oplus ss$                   $m_A \longleftarrow c_A \oplus ss$
Output: Alice's received message $m_B$ and Bob's received message $m_A$.
```

In the first step, both parties generate their public and secret keys with the corresponding public parameters. Alice randomly chooses her secret key sk_A in the keyspace $\{0, 1, ..., 2^{e_A} - 1\}$. Her public key pk_A is obtained by using the $isogen_A$ function, which can be referred to in the SIKE protocol specification document provided in [29] and so do the other three isogeny computation functions. The $isogen_A$ function for Alice is used to iteratively calculate the isogenous curves based on points P_A and Q_A with the Vélu's formulas [30] and find the images of the points Q_B and P_B on these curves. The pair of images over the isogenous curve in the last iteration is output as the public key. Analogously, Bob could get his secret and public keys in the same way by using his corresponding parameters. In the second step, They exchange their public keys and separately calculate their shared secret key j by using the $isoex_l$ ($l = A$ or B) function. The $isoex_l$ function is similar to the $isogen_l$ function, which also requires to compute the isogenous curves iteratively. The main difference lies in the initial parameters and the outputs. The $isoex_l$ function is initialized with the pair of points of the opposite party's and only needs to output the j-invariant of the last isogenous curve, computed as:

$$j = \frac{256(C^2-3)^3}{C^2-4}. \tag{2}$$

Assuming the size of plaintext is M, the j-invariant is encrypted by the hash function H to adjust the output ss with a bit width of M. Alice and Bob could securely communicate with each other by using their shared secret key as shown in Step 3: One party encrypts his/her plaintext into the ciphertext by using the XOR operation on ss and sends it to the other party, and the other party decrypts this message still using ss to do the XOR operation.

2.2 The SIKE Protocol

Note that the SIDH is proposed to resist the quantum computer's attack. However, in recent years, many other attacks [7], [8], [9] applying to specific security models have been reported. The secret keys of both parties must be dynamically updated to ensure security. To fix these flaws, authors in [29] proposed the SIKE protocol and demonstrated its security.

Algorithm 2: The SIKE protocol of PQC candidates [29].

```
Input: Public parameters: $E$, $(P_A, Q_A)$, and $(P_B, Q_B)$.
    Alice                                   Bob
  1: Generate both parties' public and secret keys, Alice's message,
     and Bob's fake message
                                            $sk_B \xleftarrow{\$} \{0,1\}^{e_B}$
                                            $pk_B \longleftarrow isogen_B(sk_B)$
     $m_A \in \{0,1\}^M$                    $fm_B \in \{0,1\}^M$
     $sk_A \longleftarrow H(\{m_A, pk_B\}, e_A)$
     $pk_A \longleftarrow isogen_A(sk_A)$
  2: Exchange the public keys and compute the shared secret key
     $j \longleftarrow isoex_A(pk_B, sk_A)$        $j \longleftarrow isoex_B(pk_A, sk_B)$
     $ss \longleftarrow H(j, M)$                   $ss \longleftarrow H(j, M)$
  3: Send the ciphertext from Alice to Bob by using the shared
     secret key and compute the output plaintext by Bob with the
     help of fake message
     $c_A \longleftarrow ss \oplus m_A$
                                            $m'_A \longleftarrow c_A \oplus ss$
                                            $sk'_A \longleftarrow H(\{m'_A, pk_B\}, e_A)$
                                            $pk'_A \longleftarrow isogen_A(sk'_A)$
     $em \leftarrow H(\{m_A, pk_A, c_A\}, K)$   $em' \leftarrow H(\{fm_B, pk_A, c_A\}, K)$
                                            $\mathrm{em}_A = \begin{cases} em, pk'_A = pk_A \\ em', pk'_A = pk_A \end{cases}$
Output: Bob's calculated message $em_A$.
```

The main course of the SIKE is shown in Alg. 2, which is assumed that Alice sends messages to Bob. We still divide this protocol into three steps. To guarantee security, some extra operations are added compared to the SIDH. In the first step, Bob generates his secret and public keys the same as in the SIDH. This secret key can be securely used repeatedly. Those keys of Alice's are produced dynamically, based on the delivered message and Bob's public key. Meanwhile, Bob produces a fake message for use in the later. The second step is the same as in the SIDH. The j-invariant is obtained by using the owner's secret key and the other party's public key and encrypted by the hash function. In the third step, Alice encrypts her message in two forms, one with their shared secret key as c_A and the other with the hash function as em, and sends them to Bob. After receiving the two ciphertexts, Bob recovers Alice's message, secret and public keys by using c_A. At the same time, he encrypts the fake message fm_B in the same way as Alice denoted by em'. He chooses em or em' as the output by judging whether Alice's recovered public key is equal to the received public key. This scheme has been proved chosen-ciphertext attack (CCA) secure.

2.3 Finite Field Arithmetic Operations for Supersingular Isogeny Elliptic Curves

The arithmetic operations over finite field mainly include the modular addition, modular subtraction, modular multiplication, modular division, modular negation, and modular inversion. The modular addition, modular subtraction, and modular negation operations are usually far simpler to be implemented compared to the rest operations. The modular division can be carried out by the modular multiplication and modular inversion operations. According to the Fermat's little theorem [31], the modular inversion operation can be computed as $A^{-1} \equiv A^{p-2} \bmod p$, also becoming the modular multiplication operations. Therefore, improving the field multiplication usually can effectively accelerate a complicated algorithm, especially in which many modular multiplications are included. The four isogeny computation functions of SIDH and SIKE exactly belong to this category.

The modular multiplication can be divided into two parts: the multiplication and modular reduction. For the multiplication part, except the schoolbook multiplication, many fast multiplication algorithms, like the Karatsuba [32], Toom-Cook [33], Schönhage–Strassen [34], and Fürer's [35] algorithms, have been proposed, which are very suitable for implementations with enough large and flexible bit widths. For the modular reduction part, two well-known modular reduction algorithms, the Montgomery reduction [25] and the Barrett reduction algorithms [27], are reported.

Since for a specific large bit width N, implementing a multiplication requires many multiplication and addition instructions, much more complex than implementing an addition or a subtraction, we will use the consumed number of multiplications to represent the complexity of an algorithm in this paper.

2.3.1 Montgomery Reduction Algorithm

The original Montgomery reduction algorithm in [25] is shown in Alg. 3, where the input operand c must satisfy

$0 \le c < Rp$ ($\gcd(p, R) = 1$) and 2 $N \times N$ multiplications are cost. Assume u digits are used to represent integers in radix-2^w, where w is set as the word size of a targeted architecture and $uw \ge N$. This algorithm requires $2u^2$ word multiplications. The multi-precision version primarily

Algorithm 3: The Montgomery reduction [25].

```
Input: An operand $c \in [0, Rp)$, the modulus $p$ with
    $2^{N-1} < p < 2^N = R$, and the pre-computed constant
    $p' = (-p^{-1}) \bmod R$.
1: $t = ((c \bmod R) \cdot p') \bmod R$
2: $r = (c + t \cdot p)/R$
3: if $r \ge p$ then
4:     $r = r - p$
5: end if
Output: The residue $r = cR^{-1} \bmod p$.
```

Algorithm 4: The multi-precision Montgomery reduction [36].

```
Input: An operand $c \in [0, Rp)$, the modulus $p$ with
    $2^{N-1} < p < 2^N \le 2^{uw} = R$, and the pre-computed
    constant $p' = (-p^{-1}) \bmod 2^w$.
1: for $i = 0$ to $u - 1$ do
2:     $t = ((c \bmod 2^w) \cdot p') \bmod 2^w$
3:     $c = (c + t \cdot p)/2^w$
4: end for
5: $r = c$
6: if $r \ge p$ then
7:     $r = r - p$
8: end if
Output: The residue $r = cR^{-1} \bmod p$.
```

proposed in [25] and improved in [36] could significantly reduce the complexity and only $(u^2 + u)$ word multiplications are needed. The multi-precision Montgomery reduction is shown in Alg. 4, where the parameter R equals 2^{uw} and in each iteration parameter R is replaced by 2^w. It should be noted that the modulus p for supersingular elliptic curves usually has the form of $p = f \cdot a^{e_A} b^{e_B} \pm 1$. If $a = 2$ and $w \le e_A$, we could have $p \bmod 2^w = \pm 1$. Thus, the parameter p' is equal to ∓ 1. The first multiplication can be removed. When w is set as the word size of a targeted architecture, we usually have $w \ll e_A$. The number of digit multiplications is reduced to u^2. The complexity can be further reduced by refining the second multiplication, which has been fully studied in [15] and [28]. The complexity in multiplication is reduced to $u(u-\delta)$, where δ is close to $u/2$. For example, when $p = 2^{372}3^{239} - 1$ and $w = 64$, we have $u = 12$ and $\delta = 5$ and the number of word multiplications is reduced from 144 to 84. Note that this method has been adopted by the SIKE protocol [17].

It should be pointed out that the output of the Montgomery reduction is not $c \bmod p$ but $cR^{-1} \bmod p$. It can be easily solved by making all involved integers in Montgomery representation, i.e., multiplying these integers by R at the beginning of an algorithm. It has been demonstrated that all arithmetic operations over a finite field can be processed in Montgomery representation the same way as in the normal representation. At the end of this algorithm, the outputs can be back to normal divided by R.

2.3.2 *Barrett Reduction Algorithm*

The primary form of the Barrett reduction algorithm [27] is presented in Alg. 5, where γ is an arbitrary integer. When the absolute value of γ is small, this algorithm needs about $3u^2$ word multiplications.

Algorithm 5: The Barrett reduction [27].

```
Input: An operand $c \in [0, 2^{2N+\gamma})$, the modulus $p$
    satisfying $2^{N-1} < p < 2^N$, and the pre-computed
    constant $\lambda = \lfloor 2^{2N+\gamma}/p \rfloor$.
1: $q = \lfloor \frac{c \cdot \lambda}{2^{2N+\gamma}} \rfloor$
2: $r = c - q \cdot p$
3: if $r \ge p$ then
4:     $r = r - p$, $q = q + 1$
5: end if
Output: The quotient $q = \lfloor c/p \rfloor$, and the remainder
    $r = c \bmod p$.
```

In [37], the authors have improved the model of the estimated quotient (Step 1 of Alg. 5) by introducing more variable parameters. The quotient is written as:

$$q = \lfloor \frac{c}{p} \rfloor = \lfloor \frac{\frac{c}{2^{N+\rho}} \frac{2^{N+\sigma}}{p}}{2^{\sigma-\rho}} \rfloor \ge \lfloor \frac{\lfloor \frac{c}{2^{N+\rho}} \rfloor \lfloor \frac{2^{N+\sigma}}{p} \rfloor}{2^{\sigma-\rho}} \rfloor, \tag{3}$$

where σ and ρ are two parameters. Assume the estimated quotient as $\hat{q} = \lfloor \frac{\lfloor \frac{c}{2^{N+\rho}} \rfloor \lfloor \frac{2^{N+\sigma}}{p} \rfloor}{2^{\sigma-\rho}} \rfloor$. The author in [38] has deduced the estimation error in detail. The final formula is:

$$q \ge \hat{q} > q - \frac{c}{2^{N+\sigma}} - \frac{2^{N+\rho}}{p} + 2^{\rho-\sigma} - 1. \tag{4}$$

Note that $0 \le c < 2^{2N+\gamma}$ and $2^{N-1} < p < 2^N$. Thus, this equation satisfies:

$$\begin{aligned} q \ge \hat{q} \quad &> \quad q - \frac{2^{2N+\gamma}}{2^{N+\sigma}} - \frac{2^{N+\rho}}{2^{N-1}} + 2^{\rho-\sigma} - 1 \\ &= \quad q - 2^{N+\gamma-\sigma} - 2^{\rho+1} + 2^{\rho-\sigma} - 1. \end{aligned} \tag{5}$$

It can be easily proved that when $\sigma \ge N + \gamma + 1$ and $\rho \le -2$, the estimation error is no larger than 1. The equal symbols both are adopted to obtain less complexity. Hence, the modified equation for the estimated quotient becomes:

$$\hat{q} = \lfloor \frac{\lfloor \frac{c}{2^{N-2}} \rfloor \lfloor \frac{2^{2N+\gamma+1}}{p} \rfloor}{2^{N+\gamma+3}} \rfloor. \tag{6}$$

When the absolute value of γ is small, this improved algorithm costs about $2u^2$ word multiplications, close to the original Montgomery reduction but still inferior to the best version.

In [26], the authors have refined the original Barrett reduction (Alg. 5) for a special structure modulus of 2^x3^y (x and y are arbitrary positive integers and $x \approx \lceil \log_2(3^y) \rceil$) and applied it as a function to their proposed reduction algorithm. In this paper, this scheme is also adopted in our new reduction algorithm that will be detailed in the next section.

We combine the methods in [26], [37], and [39], and present an improved Barrett reduction (IBR) as shown in Alg. 6. The splitting method in Step 1 is referred to [26]; the estimation method for quotient in Step 2 is referred to [37]; and the simplification methods in Steps 3, 4, and 8 are referred to [39]. The bit width of the input c is assumed as $2W+\gamma$ and the modulus is denoted as m, to distinguish from prime p. It should be noted that the IBR is not directly used as an independent reduction algorithm for the SIKE protocol but as a function to compute the quotient and remainder. It can be seen that the first multiplication (in Step 2) is a $(W+\gamma+2)\times(W+\gamma+1)$ multiplication and the second one (in Step 3) is a $(W_2+1)\times W_2$ multiplication. When the absolute value of γ is small and $W_1 \approx W_2 \approx W/2$, an IBR function takes about 1.25 $W \times W$ multiplications.

Algorithm 6: The improved Barrett reduction (IBR).

```
Input: An operand $c \in [0, 2^{2W+\gamma})$; the modulus $m = 2^x 3^y$,
    where $W_1 = x$, $W_2 = \lceil \log_2(3^y) \rceil$, $W_1 + W_2 = W$, and
    $m' = 3^y$; and the pre-computed constant
    $\lambda = \lfloor 2^{2W+\gamma+1}/m \rfloor$.
1: $t = \lfloor c/2^{W_1} \rfloor$, $s = c \bmod 2^{W_1}$
2: $q = \lfloor \frac{\lfloor \frac{t}{2^{W_2-2}} \rfloor \cdot \lambda}{2^{W+\gamma+3}} \rfloor$
3: $t_1 = (q \bmod 2^{W_2+1}) \cdot m'$
4: $r = ((t \bmod 2^{W_2+1}) - (t_1 \bmod 2^{W_2+1})) \bmod 2^{W_2+1}$
5: if $r \geq m'$ then
6:     $r = r - m'$, $q = q + 1$
7: end if
8: $r = r \cdot 2^{W_1} + s$
Output: The quotient $q = \lfloor c/m \rfloor$, and the remainder
    $r = c \bmod m$.
```

3 Proposed Data Representation for Supersingular Isogeny Based Cryptography

3.1 A Preview of the New Data Representation

Firstly, we rewrite the modulus p as:

$$\begin{aligned} p &= f \cdot a^{e_A} b^{e_B} \pm 1 \\ &= f \cdot a^{-\alpha} b^{-\beta} a^{e_A+\alpha} b^{e_B+\beta} \pm 1 \\ &= f' \cdot R^n \pm 1, \end{aligned} \tag{7}$$

where $f' = f \cdot a^{-\alpha} b^{-\beta}$, α and β are small positive integers, $n = \mathrm{GCD}(e_A+\alpha, e_B+\beta)$, $n \geq 1$, and $R = a^{\frac{e_A+\alpha}{n}} b^{\frac{e_B+\beta}{n}}$. Then, we represent a field element $A \in \mathbb{F}_p$ based on the unconventional radix R as

$$A = \sum_{j=0}^{n-1} a_j \cdot R^j, \tag{8}$$

where $a_j \in [0, R-1]$ for $0 < j < n-1$, $a_{n-1} \in [0, f'R-1]$, and $a_0 \in [0, R \pm 1]$ of which the plus or minus sign is consistent with that of p.

The correctness can be easily validated. Our goal is to build a mapping that can involve all elements in $\mathbb{F}_p$. For $p = f \cdot a^{e_A} b^{e_B} - 1$, we know that any field integer A over $\mathbb{F}_p$ satisfies $0 \leq A < p$. Meanwhile, the expression of A in Eq. (8) satisfies $0 \leq A \leq p$. Therefore, except the largest value, the values in the representation of Eq. (8) can exactly map the integers over the finite field. Equally, for $p = f \cdot a^{e_A} b^{e_B} + 1$, this new representation can also express all integers over $\mathbb{F}_p$ but is not an one-to-one mapping in some cases. Based on this new data representation, all the arithmetic operations can be correctly conducted. In the following, we will mainly demonstrate how to compute these operations based on the proposed data representation.

3.2 Deduction of A Low-Complexity Modular Reduction Algorithm

As analyzed above, the modular reduction is the core computation in the basic arithmetic operations, so we begin with it in our deduction first. Let us take two field elements $A = \sum_{j=0}^{n-1} a_j \cdot R^j$ and $B = \sum_{j=0}^{n-1} b_j \cdot R^j$, where a_j and b_j are the order-j coefficients of A and B, respectively. When multiplying the two integers, we have:

$$\begin{aligned} C &= A \times B = \sum_{j=0}^{n-1} a_j \cdot R^j \times \sum_{j=0}^{n-1} b_j \cdot R^j \\ &= \sum_{j=0}^{n-1} \sum_{i=0}^{j} a_i b_{j-i} \cdot R^j + \sum_{j=n}^{2n-2} \sum_{i=j-n+1}^{n-1} a_i b_{j-i} \cdot R^j \\ &= \sum_{j=0}^{n-1} c_j \cdot R^j + \sum_{j=n}^{2n-2} c_j \cdot R^j, \end{aligned} \tag{9}$$

where c_j for $0 \leq j \leq 2n-2$ is made up of multiply-accumulate terms based on the coefficients of A and B.

In order to make the output C with standard form as in Eq. (8) after the modulo operation, the first task is to remove the terms with orders larger than $n-1$, namely, the second multiply-accumulate term of Eq. (9). Luckily, this term can be directly merged with the first term in a general formula for the prime $p = f' \cdot R^n \pm 1$. Let us pick up an item $c_j \cdot R^j$ for $n \leq j \leq 2n-2$ and compute the modulo operation separately. According to the deduction in [28] and [23], $c_j \cdot R^j \bmod p$ satisfies

$$\begin{aligned} & c_j \cdot R^j \bmod p \\ = & ((c_j R^j \bmod f'R^n) + (\lfloor \frac{c_j R^j}{f'R^n} \rfloor \cdot f'R^n)) \bmod p \\ = & ((c_j R^j \bmod f'R^n) + (\lfloor \frac{c_j R^j}{f'R^n} \rfloor \cdot (p \mp 1))) \bmod p \\ \equiv & ((c_j R^j \bmod f'R^n) \mp \lfloor \frac{c_j R^{j-n}}{f'} \rfloor) \bmod p. \end{aligned} \tag{10}$$

For $j = n$, this formula equals:

$$\begin{aligned} & c_n \cdot R^n \bmod p \\ \equiv & (c_n R^n \bmod f'R^n) \mp \lfloor \frac{c_n}{f'} \rfloor) \bmod p \\ \equiv & ((\frac{c_n \bmod f'}{f'} \cdot f'R^n) \mp \lfloor \frac{c_n}{f'} \rfloor) \bmod p \\ \equiv & ((\frac{c_n \bmod f'}{f'} \cdot (p \mp 1)) \mp \lfloor \frac{c_n}{f'} \rfloor) \bmod p \\ \equiv & \mp(\frac{c_n \bmod f'}{f'} + \lfloor \frac{c_n}{f'} \rfloor) \bmod p. \end{aligned} \tag{11}$$

It should be pointed out that ”mod f'” is just a form to make the expression clearer. When f' is a fraction, we will not directly use it, but turn back to the previous step.

For $n < j \leq 2n-2$, we have:

$$\begin{aligned} c_j \cdot R^j \bmod p &\equiv (c_j \cdot R^n \bmod p) \cdot R^{j-n} \bmod p \quad (12) \\ &\equiv \mp\left(\frac{c_j \bmod f'}{f'} + \lfloor \frac{c_j}{f'} \rfloor\right) \cdot R^{j-n} \bmod p. \end{aligned}$$

Then, $C \bmod p$ is congruent with:

$$C \equiv c_{n-1}R^{n-1} + \sum_{j=0}^{n-2}(c_j \mp (\tfrac{c_{n+j} \bmod f'}{f'} + \lfloor \tfrac{c_{n+j}}{f'} \rfloor)) \cdot R^j \bmod p, \quad (13)$$

which is not the final output since the coefficients are usually not in the defined ranges. It should be noticed that this equation is a general formula for any p with the form of Eq. (7). For example, when $f' = 2$, $n = 2$, and $p = 2 \cdot R^2 - 1$, this result is equivalent to the derived equation of the EFFM [26]. When $f' = 1$, $n = 1$, and $p = R \pm 1$, this result is the same as that of the FFM2 [23], which has been proved less efficient than the EFFM and FFM1 in [39]. It seems that the larger n is, the faster speed can be achieved.

Intuitively, if the parameter f' is not equal to 1 or 2, the modulo and division operations over f' will become complicated. The ideal case is $f' = 1$, since $c_{n+j} \bmod f' = 0$ and the division is also removed. In that case, we have $p = R^n \pm 1$. Since the modulus p must be prime, the minus sign is not available for $n > 1$. However, in the existing SIKE examples, the prime is usually with the form of $2^{e_A}3^{e_B} - 1$ where e_A and e_B are co-prime. If directly using $f' = 1$ for the formula of Eq. (13), the modular reduction will not be so efficient since n can only be set to 1. In the following, we will demonstrate that this equation can be very simple with $n > 1$ for the primes of the SIKE.

We have found that if e_A or e_B is added by a small positive integer α or β, the revised parameters will not be co-prime and the greatest common divisor will usually be larger than 2 (referred to the parameter n in the fifth row of Table 2). Then, we have $f' = 2^{-\alpha}3^{-\beta}$. When turning back to Eq. (11), we can rewrite the first term in the first step as:

$$\begin{aligned} & c_n R^n \bmod f'R^n \quad (14) \\ = \; & c_n 2^\alpha 3^\beta \cdot f'R^n \bmod f'R^n \equiv 0. \end{aligned}$$

Therefore, we have $c_n \cdot R^n \bmod p = c_n \cdot 2^\alpha 3^\beta \bmod p$. The similar transformation can also be applied to the higher orders. Then, equation (13) becomes

$$C \equiv c_{n-1}R^{n-1} + \sum_{j=0}^{n-2}(c_j + c_{n+j}2^\alpha 3^\beta)R^j \bmod p. \quad (15)$$

3.2.1 Multiplication Part

Equation (15) can be unfolded as:

$$C \equiv \sum_{i=0}^{n-1} a_i b_{n-i-1} \cdot R^{n-1} + \sum_{j=0}^{n-2}(\sum_{i=0}^{j} a_i b_{j-i} + \sum_{i=j+1}^{n-1} a_i b_{j-i+n} \cdot 2^\alpha 3^\beta) \cdot R^j \bmod p. \quad (16)$$

Computing the raw coefficients is what the multiplication part is to do. It should be noted that these multiply-accumulate terms look like the same as those of the traditional unfolded multiplication. But their radixes are different and the bit width of a coefficient in our algorithm equals $\lceil \frac{N}{n} \rceil$. Since the coefficients are totally free from each other, our algorithm does not need to send the carry from the lower-order to the next higher-order. Additionally, as the coefficients are independent and relatively small, the one-level Karatsuba-like optimization method can be easily applied to the coefficient multiplication combination $a_ib_j + a_jb_i$ $(i \neq j)$ as:

$$a_ib_j + a_jb_i = (a_i + a_j)(b_i + b_j) - a_ib_i - a_jb_j. \quad (17)$$

Notice that the number of such combination is $(n^2 - n)/2$. By using this method, the number of coefficient multiplications is reduced from n^2 to $(n^2 - n)/2 + n = n(n+1)/2$. For the constant multiplications in Eq. (16), as the parameters, α and β, are very small (usually set to 0 or 1), only some additions and shift operations are required.

In the SIKE implementation [17], the comb multiplication algorithm [40] is adopted in the integer multiplication, which is a scheduling method to effectively control the multiplication and accumulation of partial products. As analyzed in [15], it shows more promise than the Karatsuba algorithm when the data width of operands is small or moderate. Obviously, the number of multiplication instructions is not reduced by the comb scheduling method. Besides, many carries are required to be handled in the iterations. Thus, it can believe that the proposed multiplication method could be faster than the previous one. Moreover, it should be mentioned that our multiplication method is naturally more suitable for high-parallel implementation to achieve much higher speed.

3.2.2 Low-Complexity Modular Reduction Algorithm

It is worth to mention that whatever method to compute the raw coefficients is adopted, the procedure of modular reduction for these coefficients is almost the same. Without loss of generality, we will take the prime of $2^{e_A}3^{e_B} - 1$ as an example in the following.

Suppose an integer $C = \sum_{j=0}^{n-1} c_j \cdot R^j$ with raw coefficients c_j computed by Eq. (16). The coefficients of the final output are required to be satisfied as the defined ranges, *i.e.*, $c_j \in [0, R-1]$ for $0 \leq j < n-1$ and $c_{n-1} \in [0, f'R - 1]$. The coefficients with $0 \leq j < n-1$ can be reduced by modulo R, keeping the remainders as the corresponding new coefficients and adding the quotients to the adjacent higher-order coefficients. For the $(n-1)$-th term, according to the formula

$$\begin{aligned} & c_{n-1} \cdot R^{n-1} \bmod p \quad (18) \\ \equiv \; & ((c_{n-1}R^{n-1} \bmod (f'R \cdot R^{n-1})) + \lfloor \frac{c_{n-1}R^{n-1}}{f'R^n} \rfloor) \bmod p \\ \equiv \; & ((c_{n-1} \bmod f'R) \cdot R^{n-1} + \lfloor \frac{c_{n-1}}{f'R} \rfloor) \bmod p, \end{aligned}$$

we can reduce this term by modulo $f'R$. The quotient is merged with the lowest term. Then, several additions and subtractions are needed to adjust the final result. We adopt $(n+1)$ IBR functions presented in Alg. 6 to deal with those modulo operations.

The proposed fast reduction algorithm can be summarized as follows:

1) Compute $(q_0, r_0) = \text{IBR}(c_0, R)$;

2) Calculate $(q_j, r_j) = \text{IBR}(c_j + q_{j-1}, R)$ for $0 < j < n-1$;
3) Compute $(q_{n-1}, r_{n-1}) = \text{IBR}(c_{n-1} + q_{n-2}, f'R)$;
4) Calculate $(q_0, c_0) = \text{IBR}(r_0 + q_{n-1}, R)$, where q_0 becomes very small;
5) For $0 < j \leq n-1$, compute the addition, $d_j = r_j + q_{j-1}$, and the subtraction, $e_j = d_j - R$, where R is replaced by $f'R$ for $j = n-1$. The updated quotient q_j is set as e_j's sign bit xored by 1. If q_j is equal to 0, the output c_j will be set to d_j; otherwise, c_j will be set to e_j. Finally, compute $c_0 = r_0 + q_{n-1}$.

It should be noted that the complexity of an IBR is directly related with the input c. Hence, those IBR functions can be optimized with the corresponding largest input integers. Additionally, we use a lazy reduction for c_0, with a range of $[0, R]$, to reduce the requirement of the additions and subtractions. This little trick is also adopted by the modular addition and subtraction presented later.

Now, we will try to evaluate the complexity of this new reduction algorithm in terms of multiplications. It should be noted that the multiplications only lie in the IBR functions. Obviously, The IBR of Step 4 could be designed much simpler than the others. For simplification, we use the value $((n-1)2^\alpha 3^\beta + 1)(R-1)^2$ as the largest value for all of them. The bit width equals $\log_2((n-1)2^\alpha 3^\beta + 1)(R-1)^2 = \log_2((n-1)2^\alpha 3^\beta + 1) + 2 \cdot \log_2(R-1)$. When n is relatively small, the bit width of this value is about $2 \cdot \lceil \frac{N}{n} \rceil$. For one IBR function, it costs 1.25 $\lceil \frac{N}{n} \rceil \times \lceil \frac{N}{n} \rceil$ multiplications. So, the proposed reduction algorithm takes about $1.25(n+1)$ $\lceil \frac{N}{n} \rceil \times \lceil \frac{N}{n} \rceil$ multiplications.

If $\lceil \frac{N}{n} \rceil$ is equal to w, n will be equal to u (Usually, $\lceil \frac{N}{n} \rceil$ is not exactly equal to w while we still have n equal to u. For example, for the SIKEp751, when the word size w is equal to 64, we have $n = u = 12$ but $\lceil \frac{N}{n} \rceil = \lceil \frac{751}{12} \rceil = 63$). About $1.25(n+1)$ $w \times w$ multiplications are needed. It should be pointed out that the parameter λ is assumed to be small. In that case, the ratio of the best Montgomery reduction used in SIKE library (requiring about $0.5n^2$ $w \times w$ multiplications) to our reduction is about $0.4n^2/(n+1)$ in terms of multiplications. When $n > 3$, the proposed modular reduction algorithm may obtain better performance than the fastest Montgomery one. The larger n is, the faster speed can be achieved. When targeting an architecture with specific word sizes, this trend would be limited by them. Considering the small sizes of the input operand of the IBR, the optimization methods like the Karatsuba or Toom-Cook fast multiplication can also be easily applied to the multiplications by utilizing the redundant representation. It should be pointed out that this modular reduction could be further accelerated to implement those IBR functions in parallel on software or hardware platforms.

4 Proposed Field Arithmetic Algorithms Based on the New Data Representation

The supersingular elliptic curves used for the SIDH or SIKE are usually defined over the quadratic field $\mathbb{F}_{p^2}$ which is extended from the base field $\mathbb{F}_p$ with $i^2 + 1 = 0$. Therefore, the field arithmetic operations should be considered and implemented for the two finite fields. In the following, we will propose the basic field arithmetic algorithms based on the new data representation for both fields, respectively.

4.1 Arithmetic Operations Over $\mathbb{F}_p$

4.1.1 Modular Multiplication and Squaring

As the multiplication part and the modular reduction have been presented based on the new data representation in the previous section, we can directly obtain a new modular multiplication algorithm, named as general IFFM (G-IFFM) algorithm. It can be summarized in the following two steps:

- Step 1: Compute multiply-accumulate terms in Eq. (16) and get the raw coefficients.
- Step 2: Apply the proposed low-complexity reduction algorithm to these raw coefficients and output the standard coefficients.

As analyzed above, both of the multiplication and modular reduction could be more efficient than the conventional methods. The G-IFFM can, therefore, achieve better performance than the state-of-the-art method in [17].

The modular squaring in our implementation is separately designed since Eq. (17) equals $2a_ia_j$, which need not the extra additions and subtractions, and thus is more efficient than that formula.

4.1.2 Modular Addition

For the modular addition, we can split it into two steps. Assume two field elements A and B are represented as in Eq. (8). In the first step, we directly compute

$$C = \sum_{j=0}^{n-1} (a_j + b_j) \cdot R^j = \sum_{j=0}^{n-1} c_j \cdot R^j. \tag{19}$$

Only n additions are cost without carries. In the second step, we reduce C to the standard representation. It can be noticed that we have $0 \leq c_j < 2R - 1$ for $0 \leq j < n-1$ and $0 \leq c_{n-1} < 2f'R - 1$. When $R \leq c_j < 2R - 1$ for $0 \leq j < n-1$, we can reduce this coefficient c_j by using the formula:

$$\begin{aligned} & c_{j+1} \cdot R^{j+1} + c_j \cdot R^j \\ = & (c_{j+1} + 1) \cdot R^{j+1} + (c_j - R) \cdot R^j. \end{aligned} \tag{20}$$

If the condition is satisfied, one addition and one subtraction are required.

For the coefficient c_{n-1}, we can use the following formula:

$$\begin{aligned} & c_{n-1} \cdot R^{n-1} + c_0 \bmod p \\ \equiv\ & c_{n-1} \cdot R^{n-1} - (f'R^n - 1) + c_0 \bmod p \\ =\ & (c_{n-1} - f'R) \cdot R^{n-1} + (1 + c_0) \bmod p. \end{aligned} \tag{21}$$

When satisfying $f'R \leq c_{n-1} < 2f'R - 1$, one addition and one subtraction are needed. Thus, in the worst case, the reduction for these coefficients costs n additions and n subtractions. Note that the output coefficient c_0 ranges in $[0, R]$, which does not need to be adjusted right now. In total, it takes $2n$ additions and n subtractions. To achieve constant time, some extra operations are required. More discussions will be given in the next section.

In the traditional method, two N-bit additions and one N-bit subtraction are cost by the modular addition. An N-bit addition/subtraction requires u w-bit additions/subtractions with carries/borrows, for which at least $2u$ addition/subtraction instructions are needed. Considering the overflow situation for each word addition/subtraction, it could be more complex than we count. Hence, at least $4u$ addition and $2u$ subtraction instructions are required for the modular addition. When $\lceil \frac{N}{n} \rceil \approx w$ and $n = u$, our modular addition could be at least twice faster than the traditional method.

4.1.3 Modular Subtraction

Similar to the modular addition, we can split the modular subtraction into two steps. Assume two field elements A and B are expressed as in Eq. (8). In the first step, we directly compute

$$C = \sum_{j=0}^{n-1} (a_j - b_j) \cdot R^j = \sum_{j=0}^{n-1} c_j \cdot R^j. \tag{22}$$

Only n subtractions are cost without borrows. In the second step, we reduce C to the standard representation. Notice that we have $-R+1 \le c_j \le R-1$ for $0 \le j < n-1$ and $-f'R+1 \le c_{n-1} \le f'R-1$. When $-R+1 \le c_0 \le 0$ or $-R+1 \le c_j < 0$ for $0 < j < n-1$, we can reduce them by using the formula:

$$\begin{aligned} &c_{j+1} \cdot R^{j+1} + c_j \cdot R^j \\ &= (c_{j+1} - 1) \cdot R^{j+1} + (c_j + R) \cdot R^j. \end{aligned} \tag{23}$$

For the coefficient c_{n-1} with $-f'R+1 \le c_{n-1} < 0$, we can use the following formula:

$$\begin{aligned} & c_{n-1} \cdot R^{n-1} + c_0 \bmod p \\ \equiv\ & c_{n-1} \cdot R^{n-1} + (f'R^n - 1) + c_0 \bmod p \\ =\ & (c_{n-1} + f'R) \cdot R^{n-1} + (c_0 - 1) \bmod p. \end{aligned} \tag{24}$$

Similarly, the output coefficients c_j for $0 < j \le n-1$ are in the standard ranges and c_0 falls in $[0, R]$. Our modular subtraction costs about n additions and $2n$ subtractions. In the conventional method, one N-bit addition and one N-bit subtraction are consumed for the modular subtraction. About $2u$ addition and $2u$ subtraction instructions are used. When $\lceil \frac{N}{n} \rceil \approx w$ and $n = u$, our modular subtraction could at least be 1.33 times faster than the traditional modular subtraction.

4.1.4 Modular Negation

Let C be represented as in Eq. (8). We can compute the modular negation as follows:

$$\begin{aligned} &(-C) \bmod p \equiv (P - C) \bmod p \\ &= (f'R - c_{n-1} - 1)R^{n-1} + \sum_{j=0}^{n-2} (R - c_j - 1)R^j \bmod p. \end{aligned} \tag{25}$$

It can be seen that $2n$ subtractions are required in total. In the conventional method, except the $2u$ subtractions, more operations are needed for borrows.

4.1.5 Modular Inversion

According to the Fermat's little theorem [31], the general modular inversion can be computed as $A^{-1} \equiv A^{p-2} \bmod p$, which could be calculated by using the modular multiplication and squaring operations in chains. As analyzed before, these two operations could be better than the conventional method, so the modular inversion could also obtain faster speed than the previous work.

Modular Division by Two: Besides the general case, we have also derived the modular division by two, *i.e.*, $\frac{A}{2} \bmod p$. In all cases, this modular division equals:

$$\frac{A}{2} \bmod p \equiv \begin{cases} \frac{A}{2}, & A \text{ is even}, \\ \frac{A+p}{2}, & A \text{ is odd}. \end{cases} \tag{26}$$

When $A = \sum_{j=0}^{n-1} a_j \cdot R^j$, we have $\frac{A}{2} \bmod p = \sum_{j=0}^{n-1} \frac{a_j}{2} \cdot R^j \bmod p$. We also separately compute the even and odd cases of these coefficients. If an a_j for $0 < j \le n-1$ is even, only a right shift is needed. Otherwise, we can compute it with the following decomposition:

$$\begin{aligned} \tfrac{a_j}{2} \cdot R^j &= (\frac{a_j - 1}{2} + \frac{1}{2}) \cdot R^j \\ &= (\frac{a_j - 1}{2}) \cdot R^j + \frac{R}{2} \cdot R^{j-1}. \end{aligned} \tag{27}$$

For the odd a_0, we have:

$$\frac{a_0}{2} = (\frac{a_0 - 1}{2}) + \frac{1}{2}. \tag{28}$$

The modular inversion $\frac{1}{2} \bmod p$ can be presented as:

$$\begin{aligned} &\frac{1}{2} \bmod p \equiv (\frac{p+1}{2}) \bmod p \\ &= (\frac{f'R^n}{2}) \bmod p = \frac{f'R}{2} \cdot R^{n-1} \bmod p. \end{aligned} \tag{29}$$

Therefore, our modular division by two can be summarized as follows.

First, we compute the intermediate variables b_j and c_j as:

$$b_j = \begin{cases} \frac{a_j}{2}, & a_j \text{ is even}, \\ \frac{a_j - 1}{2}, & a_j \text{ is odd}, \end{cases} \quad \text{for } 0 \le j \le n-1; \tag{30}$$

$$c_j = \begin{cases} 0, & a_{j+1} \text{ is even}, \\ \frac{R}{2}, & a_{j+1} \text{ is odd}, \end{cases} \quad \text{for } 0 \le j \le n-2, \tag{31}$$

and

$$c_{n-1} = \begin{cases} 0, & a_0 \text{ is even}, \\ \frac{f'R}{2}, & a_0 \text{ is odd}. \end{cases} \tag{32}$$

Then, the division by two is computed as:

$$\sum_{j=0}^{n-1} \frac{a_j}{2} \cdot R^j \bmod p \equiv \sum_{j=0}^{n-1} (b_j + c_j) \cdot R^j \bmod p. \tag{33}$$

The parameters $\frac{f'R}{2}$ and $\frac{R}{2}$ can be precomputed. Thus, this algorithm takes n subtractions, additions, and right shifts. The conventional method using Eq. (26) needs $2u$ additions with carries and u right-shift instructions. When $n = u$, they may cost similar number of cycles.

4.2 Arithmetic Operations Over $\mathbb{F}_{p^2}$

An element A in $\mathbb{F}_{p^2}$ is represented as $A = A_0 + A_1 i$, where $A_0, A_1 \in \mathbb{F}_p$. Assuming two elements $A, B \in \mathbb{F}_{p^2}$, the arithmetic computing over this field can be represented as:

$$\begin{aligned} A+B &= (A_0+B_0)+(A_1+B_1)i \bmod p, \qquad (34)\\ A-B &= (A_0-B_0)+(A_1-B_1)i \bmod p,\\ A\times B &= (A_0B_0-A_1B_1)+((A_0+A_1)(B_1+B_0)\\ &\quad -A_0B_0-A_1B_1)i \bmod p,\\ A^2 &= (A_0+A_1)(A_0-A_1)+(A_0A_1+A_0A_1)i \bmod p,\\ A^{-1} &= A_0(A_0^2+A_1^2)^{-1}+(-A_1(A_0^2+A_1^2)^{-1})i \bmod p, \end{aligned}$$

where the real part and imaginary part are computed separately. Each of the operations over $\mathbb{F}_{p^2}$ includes several kinds of operations over $\mathbb{F}_p$, as summarized in Table 1, where operations are in abbreviation for brevity. Since all of the involved operations over $\mathbb{F}_p$ can be better than the conventional methods, the arithmetic operations over $\mathbb{F}_{p^2}$ can also achieve better performance than the previous work. Additionally, it can be seen that the multiplication over $\mathbb{F}_p$ is widely used and dominates the computations.

TABLE 1
The Numbers of Operations in $\mathbb{F}_p$ Covered by Operations in $\mathbb{F}_{p^2}$

$\mathbb{F}_{p^2}$ \ $\mathbb{F}_p$	Add.	Sub.	Mul.	Sqr.	Inv.
Add.	2	0	0	0	0
Sub.	0	2	0	0	0
Mul.	2	3	3	0	0
Sqr.	2	1	2	0	0
Inv.	1	1	2	2	1

In fact, the multiplications, squaring, and inversion operations over $\mathbb{F}_{p^2}$ can be further optimized by separating the integer operations from the modular reductions. The integer multiplication, squaring, and addition operations can be directly used, while the integer subtraction should be specifically designed since the input of the modular reduction is nonnegative. In the following, we will mainly focus on the optimization of the modular multiplication over $\mathbb{F}_{p^2}$.

Modular Multiplication: The formula of multiplication in Eq. (34) shows that the real part contains a large subtraction. Thanks to the feature of modulo operation, the result of $(A_0B_0 - A_1B_1 = C_0 - C_1 = \sum_{j=0}^{n-1}(c_{0,j} - c_{1,j}) \cdot R^j)$ can be made positive by adding the multiple of $p = f'R^n - 1$. Thus, we can reduce it by using the formula:

$$\begin{aligned} &\sum_{j=0}^{n-1}(c_{0,j}-c_{1,j})\cdot R^j \bmod p \qquad (35)\\ \equiv &\sum_{j=0}^{n-1}(c_{0,j}-c_{1,j})\cdot R^j + xR(f'R^n-1) \bmod p\\ = &(c_{0,n-1}-c_{1,n-1}+xR(f'R-1))R^{n-1}+\\ &\sum_{j=0}^{n-2}(c_{0,j}-c_{1,j}+xR(R-1))\cdot R^j \bmod p, \end{aligned}$$

where x is a small parameter to make all of these coefficients positive. As analyzed above, the raw coefficients are no larger than $((n-1)2^\alpha 3^\beta + 1)(R-1)^2$, so x can be set close to $(n-1)2^\alpha 3^\beta$. The parameters $xR(f'R-1)$ and $xR(R-1)$ can be precomputed. We need to use extra n additions to aid this modular reduction. In the conventional method, the parameter $2^N \cdot p$ is added to help this reduction, which consumes a similar number of addition instructions but more carries.

4.3 Transformation of Data Representation

The above analyses have demonstrated that the field arithmetic operations can be normally computed based on our data representation. In fact, for a computing system, we can transform all the inputs into the new representation at the beginning, and inversely transform the final results back to normal as the output in the end. The forward and backward transformation algorithms are presented below.

4.3.1 From Normal to Unconventional Radix (N2U)

For a field element $A \in \mathbb{F}_p$, we can use Alg. 7 to transform this element into our data representation. In fact, it can be

Algorithm 7: From normal to unconventional radix (N2U).

```
Input: An operand $A \in \mathbb{F}_p$, the radix $R$, and the modulus
    $p = f'R^n - 1$.
1: for $j \leftarrow 0$ to $n-2$ do
2:     $c_j \leftarrow A \bmod R$
3:     $A \leftarrow \lfloor \frac{A}{R} \rfloor$;
4: end for
5: $c_{n-1} \leftarrow A$
Output: The result $C = \sum_{j=0}^{n-1} c_j \cdot R^j = A \bmod p$.
```

calculated by calling the IBR function with modulus R in $n-1$ times. A vector of λ with $n-1$ values is precomputed for these callings.

4.3.2 From Unconventional Radix Back to Normal (U2N)

Suppose an integer $A = \sum_{j=0}^{n-1} a_j \cdot R^j$ as defined in Eq. (8). Algorithm 8 is proposed to make the integer A from the unconventional radix back to the normal representation. Note that since the results in our representation cover the integers p and $p+1$, they should be checked out and set to zeros and ones, respectively.

5 Implementation and Benchmark Results

The publicly available implementation library of SIKE called SIDH v3.2 is widely considered as the state-of-the-art software library, which has been substantially supplemented and improved since the SIKE protocol was submitted to the NIST. The library includes four folders: *KAT* (known answer test files for the KEM), *src* (source files including C, assembly, and header files), *tests* (test files), and *Visual Studio* (Visual Studio 2015 files for compilation in Windows). In the src folder, the **generic implementations** (in portable C

Algorithm 8: From unconventional radix back to normal (U2N).

```
Input: An operand $A = \sum_{j=0}^{n-1} a_j \cdot R^j$, the radix $R$, and the
    modulus $p = f'R^n - 1$.
1: $C = a_{n-1}$
2: for $j \leftarrow n-2$ to $0$ do
3:     $C \leftarrow C \cdot R + a_j$
4: end for
5: If $C = p$, set $C$ to 0.
6: If $C = p+1$, set $C$ to 1.
Output: The result $C \in \mathbb{F}_p = A \bmod p$.
```

code) and **optimized x64 implementations** (in x64 assembly code for x64 platforms) for p434, p503, p610, and p751 are respectively provided and the **optimized ARMv8 implementations** (in ARMv8 assembly code for 64-bit ARMv8 platforms) for p503 and p751 are also covered. The difference between the three kinds of implementation lies in implementing the field arithmetic. The x64 and ARMv8 implementations both are to exploit assembly optimizations in different platforms based on the generic implementation.

In this work, we try to propose new field arithmetic algorithms to replace the old ones, aiming to speed up the whole SIKE protocol. We rely on the generic implementation software library (with no compression) and integrate the proposed field arithmetic functions into it to show a more complete picture of the SIKE protocol acceleration brought by the new techniques. The optimization for the x64 or ARM platform is not considered in this paper.

5.1 Parameters Breakdown for the SIKE Protocol

The four groups of primes for SIKE all have the form of $2^{e_A}3^{e_B}-1$ with coprime factors e_A and e_B. According to our aforementioned method, they can be easily broken down with the parameters listed in Table 2. The bit widths of the obtained unconventional radices are appended in the fourth row. The digits u required in [17] are added in the fifth row. It can be seen that when the unconventional radix R is larger than $2^6 4$, the polynomial order n is usually smaller than u. For example, for SIKEp434, $n = 6$ but $u = 7$; for SIKEp610, $n = 6$ but $u = 10$. Though the word length of a coefficient increases to 2, the order is reduced to some degree. However, those cases for complexity estimation are uncertain. So we just take a certain case ($n = u$) to show the trend in the above sections. The parameters in the table can also be changed by using different values of f'.

The parameters of SIKEp751 are selected as an example to show the efficiency of the proposed field arithmetic in this paper. The overall results of the SIKEp751 and SIKEp503 will also be added as supplemental experiments. We coded our design in C language and benchmarked it on an Intel Xeon E5-2690 processor with a 64-bit operating system. The generic implementation in C code of the SIKE library [17] was also run on this processor for a fair comparison. The TurboBoost was disabled during all the tests. Our code is available at: https://github.com/FastSIKE2019/generic[1].

5.2 Analysis of Finite Field Arithmetic Computing

As introduced in Section 2, the basic arithmetic operations are the cornerstones of the SIKE protocol. We have counted those operations and calculated the proportions of clock cycles of them in the SIKEp751 for our and the previous implementations, respectively. Some of them are selected and listed in Table 3. The running time will be reported in the following. It should be noted that the numbers of operations are close to but not the same in many cases between the two implementations because of the adopted different methods. It can be seen that the modular reduction and the integer multiplication operations have a dominant position in both libraries. Optimizing the two operations is very effective to accelerate the whole protocol. Note that the modular reduction and the integer multiplication are not merely used in the modular multiplications as analyzed in the previous section. The modular addition and subtraction operations over $\mathbb{F}_p$ take up about half of the rest of the proportion. The other operations, like the hash function and the modular negation, are trivial for the whole system. Meanwhile, we can find that the proportion of modular reduction is reduced in our work while that of the multiplication goes up. This is because we have achieved more simplification on the former than the latter. More results will be provided in the following to explain this phenomenon.

Table 4 shows the running time (average clock cycles) of operations over the selected base field and its quadratic field, where the acceleration factors (AF) are also given in the right-most column. It can be seen that all of these basic arithmetic operations of our work achieve faster speed than those in [17] over either field.

5.2.1 Impact of the Optimized Modular Multiplication Over $\mathbb{F}_p$

The multiplication over $\mathbb{F}_p$ is composed of modular reduction and integer multiplication. For the reduction part, based on our analysis in Section 3, by using the proposed reduction algorithm, about 77% reduction in multiplications would be obtained. However, we cannot take full advantage of every bit. The computations in software usually are predefined as instructions with fixed word sizes. Take the proposed modular reduction algorithm for an example, compared with the Montgomery one proposed in [17], benchmarked on a 64-bit operating system for the SIKEp751. We will analyze the numbers of required multiplication instructions of them, respectively. In [17], the *MUL* function (a digital multiplication) is called $12 \times 7 = 84$ times and thus $84 \times 4 = 336$ multiplication instructions are needed. In our algorithm, the IBR function is called $12 + 1 = 13$ times. The maximum data width of the input c of the IBR is 132 ($2 \times 63 + 6$) and the sizes of the two multiplications are 72×72 and 32×32, respectively. For the first multiplication, we divide the inputs into three digits and use 6

1. We can currently only provide the .o files of the field arithmetic, which is sufficient to reproduce the execution time of our implementation when using the same target platform. We will upload the rest c files as soon as possible.

TABLE 2
Breaking Down the Parameters of the Primes Provided in [17] with Our Method

Prime	SIKEp434	SIKEp503	SIKEp610	**SIKEp751**
Security	Level 1	Level 2	Level 3	Level 5
Form	$2^{216}3^{137}-1$	$2^{250}3^{159}-1$	$2^{305}3^{192}-1$	$2^{372}3^{239}-1$
R	$2^{36}3^{23}$ (73 bits)	$2^{25}3^{16}$ (51 bits)	$2^{51}3^{32}$ (102 bits)	$2^{31}3^{20}$ (63 bits)
n/u	6/7	10/8	6/10	12/12
f'	$\frac{1}{3}(\alpha=0,\ \beta=1)$	$\frac{1}{3}(\alpha=0,\ \beta=1)$	$\frac{1}{2}(\alpha=1,\ \beta=0)$	$\frac{1}{3}(\alpha=0,\ \beta=1)$

TABLE 3
The Statistics of Selected Basic Arithmetic Operations in the SIKEp751 Library

Operation	Number of Operation		Proportion in Protocol	
	[17]	Our work	[17]	Our work
Reduc.	234,209	234,175	32.23%	20.18%
Mul.	307,888	307,986	61.05%	76.88%
Add.	87,814	168,310	1.77%	1.01%
Sub.	130,638	130,638	1.68%	0.76%

multiplication instructions to implement it. Thus, 7 multiplication instructions are adopted for one IBR, and therefore, $13\times7=91$ multiplication instructions are consumed. About 73% of the multiplication instructions are reduced by our method. It should be noted that the other instructions (like the addition, subtraction, or shift) may not have such much reduction. In brief, the proposed reduction algorithm saves more than 60% cycles (*i.e.*, about 2.61x speedup shown in Table 4) compared to the one used in [17].

For the integer multiplication part, the proposed algorithm has two aspects of optimization to reduce the computation complexity, compared to the multi-precision comb multiplication algorithm used in [17]. On one hand, there are no carries to be propagated in the adjacent orders; on the other hand, the coefficient multiplication terms $a_ib_j+a_jb_i$ ($i\neq j$) can be easily simplified. For the SIKEp751, the bit width of a coefficient is no larger than 63, we can simply use the one-level Karatsuba-like method to reduce the complexity. With this help, the number of calling the 64×64 multiplication function is reduced from 144 to 78. According to Eq. (16), more additions are required to merge the higher-order terms with the corresponding lower-order terms. Meanwhile, the higher-order terms need to be multiplied with a small constant. For the SIKEp751, this constant is equal to 3. The constant multiplication can be replaced by a 1-bit left-shift and an addition. Hence, the speedup of the multiplication is cut down, only a factor of 1.32.

As shown in Table 4, combining the modular reduction with integer multiplication, the proposed modular multiplication over $\mathbb{F}_p$ is 1.65x superior to the previous implementation.

The modular squaring is further optimized based on the modular multiplication with a speedup of 2.26x. Since the modular inversion is made up of the modular multiplications and squaring, this operation also has a factor of 2.14x speedup.

5.2.2 *Impact of the Optimized Modular Addition and Subtraction Over $\mathbb{F}_p$*

From Table 4, we can see that the AFs of modular addition and subtraction both are drastically larger than the ratios estimated in Section 4. It means that the generic version in [17] consumes many more extra operations to handle the carry or borrow. Those may be greatly simplified implemented on an x64 or ARM platform in assembly code. Nevertheless, it can still believe that the proposed modular addition/subtraction can outperform the previous one when running on the same platform. Additionally, this attenuation in acceleration here would not affect the whole SIKE implementation so significantly as the running time of modular addition is not the bottleneck according to the statistics in Table 3.

5.2.3 *Impact of the Optimized Operations Over $\mathbb{F}_{p^2}$*

The operations over $\mathbb{F}_{p^2}$ are mainly constituted by the corresponding operations over $\mathbb{F}_p$ as shown in Table 1. It can be observed that except the modular squaring, the other operations show a similar trend of a speedup as the corresponding operations over $\mathbb{F}_p$. That is because the modular squaring over $\mathbb{F}_{p^2}$ is mainly decomposed into not modular squaring but multiplication operations over $\mathbb{F}_p$.

5.3 Performance comparison of the SIKE Protocol

The overall comparison results of the three phases (KeyGen, Encaps, Decaps) for SIKEp751 are shown in Table 5. In all cases, we can achieve about 1.65x speedup. The total design is also about 1.65x faster than the method implemented in the SIDH v3.2 library. Note that these AF values are very close to the AF values of modular multiplication over $\mathbb{F}_p$ (1.65x). This result, in return, demonstrates the dominant position of the modular multiplication. We have also implemented the software for SIKEp503 and tested the running time as shown in Table 6. It can be seen that our implementation obtains an about 1.61x speedup over the previous work, which further demonstrates the effectiveness of our method.

6 Conclusions

In this paper, we have presented a faster software implementation of the SIKE protocol based on our proposed data representation. This new data representation is a general form for the supersingular isogeny-based elliptic curves, which can facilitate faster finite field arithmetic computing than prior arts. With the help of this representation, we have derived a low-complexity modular reduction algorithm for the prime of $p=2^{e_A}3^{e_B}-1$, which is usually considered in the SIKE implementation. Besides, the other basic field arithmetic algorithms are deduced and discussed. We have applied these proposed algorithms to the SIKE protocol and successfully validated all of them. When benchmarked on an Intel Xeon E5-2690 processor and compared with the state-of-the-art software implementations, the new SIKE implementation achieves 1.65x and 1.61x speedup for the

TABLE 4
Timing Performance of Selected Base Field and Quadratic Field Operations of SIKEp751. Timings Are Reported in Clock Cycles

Field	Operation	[17]	Our work	AF
$\mathbb{F}_p$	Mul. (Reduc. & Int.Mul.)	4808 (1981 & 2827)	2906 (760 & 2146)	1.65 (2.61 & 1.32)
	Sqr.	4997	2207	2.26
	Add.	286	52	5.50
	Sub.	191	48	3.98
	Inv.	4,490,413	2,097,965	2.14
$\mathbb{F}_{p^2}$	Mul.	13141	8288	1.59
	Sqr.	10047	5933	1.69
	Add.	572	103	5.55
	Sub.	389	91	4.27

TABLE 5
Overall Timing Comparisons of the SIKEp751 Software Implementations. Timings Are Reported in Clock Cycles

Phase	[17]	Our work	AF
KeyGen	330,394,357	200,167,938	1.651
Encaps	535,098,458	324,778,282	1.648
Decaps	575,180,241	348,305,883	1.651
Total	1,440,673,056	873,252,103	1.650

TABLE 6
Overall Timing Comparisons of the SIKEp503 Software Implementations. Timings Are Reported in Clock Cycles

Phase	[17]	Our work	AF
KeyGen	99,448,697	61,837,086	1.608
Encaps	163,759,088	101,847,565	1.608
Decaps	174,201,386	108,200,191	1.610
Total	437,409,171	271,884,842	1.609

SIKEp751 and the SIKEp503, respectively. It should be noted that higher acceleration factors are obtained for most of the proposed field operations.

Though these improvements are significant, it still has a big gap between the SIKE protocol and some other popular candidates. As analyzed above, most of the proposed algorithms for the SIKE are very suitable for high parallel design, such as the integer multiplication, modular addition, modular subtraction, and modular negation, thanks to the independent coefficients computing. This is completely different from conventional methods. When fully adopting the parallelism strategy, the running time of the new implementation is very likely to be accelerated in multiples, perhaps with a factor of n. It could be very interesting and meaningful to do such exploration in software or hardware. Our future work will mainly focus on this point to further bridge the gap.

References

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[2] V. S. Miller, "Use of elliptic curves in cryptography," in *Conference on the theory and application of cryptographic techniques*. Springer, 1985, pp. 417–426.

[3] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[4] L. Chen, L. Chen, S. Jordan, Y.-K. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone, *Report on post-quantum cryptography*. US Department of Commerce, National Institute of Standards and Technology, 2016, vol. 12.

[5] R. Azarderakhsh, M. Campagna, C. Costello, L. Feo, B. Hess, A. Jalali, D. Jao, B. Koziel, B. LaMacchia, P. Longa *et al.*, "Supersingular isogeny key encapsulation," *Submission to the NIST Post-Quantum Standardization project*, 2017.

[6] D. Hofheinz, K. Hövelmanns, and E. Kiltz, "A modular analysis of the fujisaki-okamoto transformation," in *Theory of Cryptography Conference*. Springer, 2017, pp. 341–371.

[7] S. D. Galbraith, C. Petit, B. Shani, and Y. B. Ti, "On the security of supersingular isogeny cryptosystems," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2016, pp. 63–91.

[8] A. Gélin and B. Wesolowski, "Loop-abort faults on supersingular isogeny cryptosystems," in *International Workshop on Post-Quantum Cryptography*. Springer, 2017, pp. 93–106.

[9] Y. B. Ti, "Fault attack on supersingular isogeny cryptosystems," in *International Workshop on Post-Quantum Cryptography*. Springer, 2017, pp. 107–122.

[10] D. Jao and L. De Feo, "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies," in *International Workshop on Post-Quantum Cryptography*. Springer, 2011, pp. 19–34.

[11] D. Jao, "Software for "towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies"," 2011, https://github.com/defeo/ss-isogeny-software.

[12] R. Azarderakhsh, D. Fishbein, and D. Jao, "Efficient implementations of a quantum-resistant key-exchange protocol on embedded systems," *Citeseer*, 2014.

[13] R. Azarderakhsh, D. Jao, K. Kalach, B. Koziel, and C. Leonardi, "Key compression for isogeny-based cryptosystems," in *Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography*. ACM, 2016, pp. 1–10.

[14] C. Costello, P. Longa, and M. Naehrig, "Efficient algorithms for supersingular isogeny Diffie-Hellman," in *Annual International Cryptology Conference*. Springer, 2016, pp. 572–601.

[15] A. Faz-Hernández, J. López, E. Ochoa-Jiménez, and F. Rodríguez-Henríquez, "A faster software implementation of the supersingular isogeny Diffie-Hellman key exchange protocol," *IEEE Transactions on Computers*, vol. 67, no. 11, pp. 1622–1636, 2017.

[16] G. H. Zanon, M. A. Simplicio, G. C. Pereira, J. Doliskani, and P. S. Barreto, "Faster key compression for isogeny-based cryptosystems," *IEEE Transactions on Computers*, vol. 68, no. 5, pp. 688–701, 2018.

[17] C. Costello, P. Longa, and M. Naehrig, "PQCrypto-SIDH," 2019, https://github.com/Microsoft/PQCrypto-SIDH.

[18] B. Koziel, R. Azarderakhsh, M. M. Kermani, and D. Jao, "Post-quantum cryptography on FPGA based on isogenies on elliptic curves," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 1, pp. 86–99, 2017.

[19] B. Koziel, R. Azarderakhsh, and M. M. Kermani, "A high-performance and scalable hardware architecture for isogeny-based cryptography," *IEEE Transactions on Computers*, vol. 67, no. 11, pp. 1594–1609, 2018.

[20] H. Seo, Z. Liu, P. Longa, and Z. Hu, "SIDH on ARM: faster modular multiplications for faster post-quantum supersingular isogeny key exchange," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 1–20, 2018.

[21] A. Jalali, R. Azarderakhsh, and M. M. Kermani, “NEON SIKE: supersingular isogeny key encapsulation on ARMv7,” in *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer, 2018, pp. 37–51.

[22] A. Jalali, R. Azarderakhsh, M. M. Kermani, and D. Jao, “Towards optimized and constant-time CSIDH on embedded devices,” in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2019, pp. 215–231.

[23] W. Liu, J. Ni, Z. Liu, C. Liu, and M. O’Neill, “Optimized modular multiplication for supersingular isogeny Diffie-Hellman,” *IEEE Transactions on Computers*, pp. 1–1, 2019.

[24] B. Koziel, A.-B. Ackie, R. E. Khatib, R. Azarderakhsh, and M. Mozaffari-Kermani, “SIKE’d Up: Fast and secure hardware architectures for supersingular isogeny key encapsulation,” Cryptology ePrint Archive, Report 2019/711, 2019, https://eprint.iacr.org/2019/711.

[25] P. L. Montgomery, “Modular multiplication without trial division,” *Mathematics of computation*, vol. 44, no. 170, pp. 519–521, 1985.

[26] A. Karmakar, S. S. Roy, F. Vercauteren, and I. Verbauwhede, “Efficient finite field multiplication for isogeny based post quantum cryptography,” in *International Workshop on the Arithmetic of Finite Fields*. Springer, 2016, pp. 193–207.

[27] P. Barrett, “Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor,” in *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 1986, pp. 311–323.

[28] J. Bos and S. Friedberger, “Arithmetic considerations for isogeny based cryptography,” *IEEE Transactions on Computers*, pp. 1–1, 2018.

[29] D. Jao *et al.*, “Supersingular isogeny key encapsulation (SIKE),” *Submission to NIST Post-Quantum Cryptography Standardization*, 2017.

[30] J. Vélu, “Isogénies entre courbes elliptiques,” *CR Acad. Sci. Paris, Séries A*, vol. 273, pp. 305–347, 1971.

[31] E. W. Weisstein, “Fermat’s little theorem,” *From MathWorld–A Wolfram Web Resource*, 2004, https://mathworld.wolfram.com/FermatsLittleTheorem.html.

[32] A. A. Karatsuba and Y. P. Ofman, “Multiplication of many-digital numbers by automatic computers,” in *Doklady Akademii Nauk*, vol. 145, no. 2. Russian Academy of Sciences, 1962, pp. 293–294.

[33] S. A. Cook and S. O. Aanderaa, “On the minimum computation time of functions,” *Transactions of the American Mathematical Society*, vol. 142, pp. 291–314, 1969.

[34] A. Schönhage and V. Strassen, “Schnelle multiplikation grosser zahlen,” *Computing*, vol. 7, no. 3-4, pp. 281–292, 1971.

[35] M. Fürer, “Faster integer multiplication,” *SIAM Journal on Computing*, vol. 39, no. 3, pp. 979–1005, 2009.

[36] S. R. Dussé and B. S. Kaliski, “A cryptographic library for the motorola DSP56000,” in *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1990, pp. 230–244.

[37] J.-F. Dhem and J.-J. Quisquater, “Recent results on modular multiplications for smart cards,” in *International Conference on Smart Card Research and Advanced Applications*. Springer, 1998, pp. 336–352.

[38] Y. Kong, “Optimizing the improved Barrett modular multipliers for public-key cryptography,” in *2010 International Conference on Computational Intelligence and Software Engineering*, Dec 2010, pp. 1–4.

[39] J. Tian, J. Lin, and Z. Wang, “Ultra-fast modular multiplication implementation for isogeny-based post-quantum cryptography,” in *2019 IEEE Workshop on Signal Processing Systems (SiPS)*, 2019.

[40] P. G. Comba, “Exponentiation cryptosystems on the IBM PC,” *IBM systems journal*, vol. 29, no. 4, pp. 526–538, 1990.