Efficient Software Implementation of the SIKE Protocol Using a New Data Representation

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

Abstract—Thanks to relatively small public and secret keys, the Supersingular Isogeny Key Encapsulation (SIKE) protocol made it into the third evaluation round of the post-quantum standardization project of the National Institute of Standards and Technology (NIST). Even though a large body of research has been devoted to the efficient implementation of SIKE, its latency is still undesirably long for many real-world applications. Most existing implementations of the SIKE protocol use the Montgomery representation for the underlying field arithmetic since the corresponding reduction algorithm is considered the fastest method for performing multiple-precision modular reduction. In this paper, we propose a new data representation for supersingular isogeny-based Elliptic-Curve Cryptography (ECC), of which SIKE is a sub-class. This new representation enables significantly faster implementations of modular reduction than the Montgomery reduction, and also other finite-field arithmetic operations used in ECC can benefit from our data representation. We implemented all arithmetic operations in C using the proposed representation such that they have constant execution time and integrated them to the latest version of the SIKE software library. Using four different parameters sets, we benchmarked our design and the optimized generic implementation on a 2.6 GHz Intel Xeon E5-2690 processor. Our results show that, for the prime of SIKEp751, the proposed reduction algorithm is approximately 2.61 times faster than the currently best implementation of Montgomery reduction, and our representation also enables significantly better timings for other finite-field operations. Due to these improvements, we were able to achieve a speed-up by a factor of about 1.65, 2.03, 1.61, and 1.48 for SIKEp751, SIKEp610, SIKEp503, and SIKEp434, respectively, compared to state-of-the-art generic implementations.

Index Terms—Supersingular Isogeny Diffie-Hellman (SIDH) key exchange, Elliptic Curve Cryptography (ECC), modular reduction operation, Montgomery representation, Barrett representation, Post-Quantum Cryptography (PQC).

1 Introduction

Due to recent progress in building quantum circuits, a large-scale quantum computer might become reality within the next 10 to 20 years. The hard mathematical problems underpinning the security of widely-used public-key cryptosystems like the Rivest-Shamir-Adleman (RSA) algorithm [1] and Elliptic-Curve Cryptography (ECC) [2] could be easily solved by using Shor's algorithm [3] on a large-scale quantum computer. These recent developments have, in turn, accelerated the establishment of Post-Quantum Cryptography (PQC), a relatively new sub-area or cryptography concerned with the design and analysis of cryptographic algorithms that will remain secure in the dawning era of quantum computing. In 2017, the National Institute of Standards and Technology (NIST) [4] has started an initiative to evaluate and standardize post-quantum cryptographic algorithms. The Supersingular Isogeny Key Encapsulation (SIKE) protocol [5] is an alternate candidate for public-key encryption and key-establishment in the third round of the evaluation process. This protocol has the advantage of relatively short public and secret key sizes, making it highly compatible with conventional ECC protocols like ECDH key exchange. SIKE is based on the Supersingular Isogeny Diffie-Hellman (SIDH) key exchange, which was proposed by Jao and De Feo in 2011 [6] as a post-quantum cryptosystem whose security rests on the difficulty of finding isogenies between supersingular elliptic curves. In essence, SIKE applies a Fujisaki-Okamoto transformation from [7] on SIDH to obtain a Key-Encapsulation Mechanism (KEM) that is secure against Chosen Ciphertext Attacks (CCA) and shows increased robustness against physical attacks [8], [9], [10]. State-of-the-art parameters for SIKE use supersingular curves over prime fields of a length of between 434 and 751 bits. The main drawback of SIKE is high computational cost and long latency, caused mainly by the serial computation of these isogenies, which forms a serious bottleneck for practical applications. Therefore, techniques to accelerate SIDH and SIKE are an important topic in PQC research.

Over the past few years, many researchers have devised optimizations for the SIDH/SIKE protocol for both software [11], [12], [13], [14], [15], [16], [17] and hardware [18], [19], [20], [21], [22], [23], [24] platforms. The very first implementation of SIDH was introduced almost 10 years ago by Jao [11]. It uses the GMP library for the field arithmetic and has served as starting point and reference for subsequent developments. The latest version provided in [17] is commonly recognized as the fastest software implementation and supports many state-of-the-art supersingular isogeny cryptographic schemes. In addition, the library in [17] covers implementations of SIKE on Intel x64, ARM, and FPGA platforms, which also combine most optimization techniques presented in the open literature. In recent years, various improvements have been proposed

- *J. Tian, P. Wang, J. Lin, and Z. Wang are with the School of Electronic Science and Engineering, Nanjing University, China. Email: {tianjing, jlin, zfwang}@nju.edu.cn, piaoyang_wang@smail.nju.edu.cn*
- *Z. Liu is with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China. Email: zhe.liu@nuaa.edu.cn*
- *J. Großschädl is with the University of Luxembourg, L–4365 Esch-sur-Alzette, Luxembourg. Email: johann.groszschaedl@uni.lu*

(Corresponding author: Jun Lin and Zhongfeng Wang.)

to speed up the SIKE protocol and make it more practical. However, all these implementations are still more than one order of magnitude slower than alternative PQC candidates. It should be noted that almost all existing implementations are based on the Montgomery representation for the field arithmetic. The main reason is that the associated reduction algorithm [25] is generally regarded as the most efficient technique for performing modular reduction. Furthermore, the Montgomery representation also allows efficient implementation of other field-arithmetic operations.

The finite fields used in supersingluar isogeny elliptic curve cryptography are quadratic extension fields of a prime field $\mathbb{F}_p$ given by a prime of the form $p = f \cdot a^{e_A} b^{e_B} \pm 1$, where a and b are small primes (e.g., $a = 2$, $b = 3$), e_A and e_B are positive integers, and f is a small cofactor. Considering the special structure of these primes, various research efforts have been made to improve the efficiency of the reduction modulo p. In [26], Karmakar *et al.* proposed a modular reduction technique for primes 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 an efficient formula to replace the modulo-p operations by two modulo-R operations. The implementation of these two reduction operations was inspired by Barrett's technique [27] and involves a special division algorithm. According to their experiments, the new reduction method achieved better performance than the original Barrett technique and also outperformed Montgomery's reduction method. Liu *et al.* presented in [23] two improved variants of Barrett reduction based on the unconventional quadratic representation from [26] and demonstrated their efficiency for hardware implementation. Also Bos *et al.* [28] analyzed and compared techniques to perform efficient arithmetic modulo the special primes used in SIKE and came to the conclusion that approaches based on Montgomery reduction are to be preferred.

Our contributions: In this paper, we further explore the data representation based on an unconventional radix for the implementation of supersingular isogeny elliptic curve cryptosystems. We transform the conventional structure of primes and extend the orders from quadratic to any orders. Based on this representation, we propose a low-complexity method for modular reduction according to the improved Barrett reduction algorithm. This new method shows a great potential to outperform the best Montgomery reduction algorithm. Furthermore, our special data representation is also applicable to other field-arithmetic operations. We implemented the proposed arithmetic algorithms in software and integrated them to the SIKE protocol. The entire implementation has constant execution time in order to withstand timing attacks.

The main contributions are summarized as follows:

1) A general data representation of field elements is proposed for supersingular isogeny elliptic curves, which can facilitate faster computations of the field arithmetic operations.
2) An efficient modular reduction algorithm is derived based on the proposed data representation with several novel ideas. This new reduction method achieves a significant complexity reduction and shows clear superiority over the fastest Montgomery reduction algorithm when adopting a high-order representation.
3) Other field operations are also deduced based on the proposed data representation. The operations on the new form require fewer computations than the 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 2.6 GHz Intel Xeon E5-2690 processor and set the order of representation to 12. All tests of these algorithms using the parameters for SIKEp751 (NIST security level 5) passed successfully. The proposed modular reduction algorithm achieves a 2.61-fold speed-up over the fastest Montgomery reduction algorithm and also the other tested field-arithmetic operations are computed faster than those in previous works. In addition, the full implementation for SIKEp751 is roughly 1.65 times faster 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 the basic field-arithmetic operations. The proposed data representation and the efficient modular reduction algorithm are presented in Section 3. The finite-field arithmetic algorithms based on the new data representation are derived in Section 4. In Section 5, experimental results and comparisons are provided. Finally, Section 6 concludes the paper.

2 Preliminaries

As mentioned in the last section, the SIKE protocol is a CCA-secure KEM based on the SIDH key exchange of Jao and De Feo [11]. By analyzing the operations of this protocol, it can be easily found that the large-degree isogeny computations dominate the overall execution time. Those computations over elliptic curves can be divided into the basic arithmetic operations in a quadratic extension field $\mathbb{F}_{p^2}$, where p is a prime of the form $p = f \cdot a^{e_A} b^{e_B} \pm 1$. The arithmetic in $\mathbb{F}_{p^2}$ can be further subdivided into arithmetic operations in the prime field $\mathbb{F}_p$. Usually, arithmetic in $\mathbb{F}_p$ is implemented using the Montgomery representation [25] since the associated reduction algorithm is considered to have the best performance.

2.1 The SIDH Protocol

The SIDH key-exchange protocol allows two parties, e.g. Alice and Bob, to securely communicate with each other based on a shared secret key negotiated through public communication. This shared secret is the j-invariant of two isomorphic supersingular elliptic curves obtained from a public supersingular elliptic curve E. The main steps of this protocol are summarized in Alg. 1, where the operations in the left column are computed by Alice and those in the right column by Bob. The public supersingular elliptic curve is usually given as a Montgomery curve of the form

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

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 $\{P_A, Q_A\}$

Algorithm 1: The SIDH key-exchange protocol [6].

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.

and $\{P_B, Q_B\}$ are all 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.

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 is described in detail in the SIKE protocol specification document [29] along with the other three isogeny computation functions. The $isogen_A$ function for Alice is used to iteratively calculate isogenous curves based on points P_A and Q_A via Vélu's formulas [30] and find the images of points Q_B and P_B on these curves. The pair of images over the isogenous curve in the last iteration is output as public key. Analogously, Bob gets his secret and public keys in the same way by using his corresponding parameters. In the second step, they exchange their public keys and calculate their shared secret key j separately with the $isoex_l$ ($l = A$ or B) function. The $isoex_l$ function is similar to $isogen_l$ and also requires to compute isogenous curves iteratively. However, they differ in their initial parameters and the outputs. The $isoex_l$ function is initialized with the pair of points of the opposite party 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 plaintext is M bits long, the shared secret ss is derived from the j-invariant with the help of a hash function H in such a way that ss has a length of M bits. Alice and Bob can now securely communicate with each other thanks to their shared secret key as shown in Step 3: One party encrypts a plaintext into a ciphertext by XORing it with ss and sends it to the other party. The other party recovers the plaintext by applying the XOR operation on the ciphertext and ss.

2.2 The SIKE Protocol

Though SIDH is able to resist quantum cryptanalysis, it becomes insecure when one party reuses its secret key and is also a relatively easy target for physical attacks [8], [9], [10]. SIKE overcomes these weaknesses with the help of a variant of the Fujisaki-Okamoto transform [7], which turns the original SIDH key exchange protocol into a CCA-secure KEM protocol [29]. Both protocols are very similar from an arithmetic point of view, but only SIKE remains secure with static keys and provides indistinguishability in the CCA scenario.

Algorithm 2: The SIKE protocol as specified in [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**	
	$c_A \longleftarrow ss \oplus m_A$	
		$m'_A \longleftarrow c_A \oplus ss$
		$sk'_A \longleftarrow H(\{m'_A, pk_B\}, e_A)$
		$pk'_A \longleftarrow isogen_A(sk'_A)$
	$sm \leftarrow H(\{m_A, pk_A, c_A\}, K)$	$sm' \leftarrow H(\{m'_A, pk'_A, c_A\}, K)$
		$sm'_B \leftarrow H(\{fm_B, pk_A, c_A\}, K)$

$$sm_{AB} = \begin{cases} sm', pk'_A = pk_A \\ sm'_B, pk'_A \neq pk_A \end{cases}$$

Output: Bob's calculated message sm_{AB}.

The main course of SIKE is specified in Alg. 2, whereby it is assumed that Alice sends a message to Bob. Similarly as before, we divide this protocol into three main steps, but in order to guarantee security, some extra operations are added compared to SIDH. In the first step, Bob generates his secret and public key in the same way as in SIDH (but contrary to SIDH, the secret key can be securely used repeatedly). The keys of Alice are produced dynamically, based on the delivered message and Bob's public key. Furthermore, Bob generates a fake message for use in the third step. The second step is exactly the same as in SIDH, namely the j-invariant is obtained by a computation that uses one's own secret key and the other party's public key as input. Both Alice and Bob hash the j-invariant to get the shared secret ss. Then, in the third step, Alice encrypts her message in two ways, namely once with the shared secret ss as c_A and the second time with the hash output $H(\{m_A, pk_A, c_A\}, K)$ as sm, where K represents the number of (classical) bits of security. She keeps sm as their new shared key and sends the ciphertext c_A to Bob. After receiving c_A, Bob recovers Alice's message and computes her secret and public key, respectively. Then, he encrypts the recovered message m'_A and the fake message fm_B in the same way as Alice to obtain sm' and sm'_B, respectively. He chooses sm' or sm'_B as the output by judging whether Alice's recovered public key is equal to the received public key. This protocol has been proven CCA secure and can be protected effectively against side-channel attacks as demonstrated by Zhang *et al.* in [31].

2.3 Finite-Field Arithmetic Operations for SIDH/SIKE

The finite-field operations that are most relevant for SIDH and SIKE are modular addition, modular subtraction, modular negation, modular multiplication, modular division, and modular inversion. Among these operations, the first three are usually far simpler to implement than the latter three. The modular division can be composed of a modular multiplication and a modular inversion. According to Fermat's little theorem [32], a modular inversion can be carried out by computing $A^{-1} \equiv A^{p-2} \bmod p$, which boils down to a sequence of modular multiplications. Therefore, improving the execution time of the field multiplication is an effective way to accelerate more complicated operations, especially when they involve many multiplications. The four *isogen*-functions of SIDH and SIKE exactly belong to this category.

A modular multiplication can be divided into two parts: multiplication and modular reduction. For the multiplication part, a number of algorithms with lower complexity than the schoolbook method have been proposed, e.g. the algorithm of Karatsuba [33], Toom-Cook [34], Schönhage-Strassen [35], and Fürer [36], which are suitable for implementation when the operands are sufficiently large. On the other hand, for the modular reduction operation, there exist two well-known algorithms, namely the Montgomery reduction [25] and Barrett's reduction technique [27].

Since implementing a multiplication for operands of a large bit-length N requires many multiply and add instructions (much more than for a modular addition or subtraction), we will mainly use the number of executed multiply instructions to analyze the complexity of an algorithm.

2.3.1 Montgomery Reduction Algorithm

The original Montgomery reduction algorithm [25] is shown in Alg. 3, whereby the input operand c to be reduced has to satisfy $0 \leq c < Rp$ and the Montgomery radix R must be relatively prime to p, i.e. $\gcd(p, R) = 1$. When implemented on a processor with a word-size of w bits, the prime p consists of $u = \lceil N/w \rceil$ digits and we have $uw \geq N$. Computing the reduction as shown in Alg. 3 requires $2u^2$ digit-multiplications.

Algorithm 3: Montgomery reduction [25].

Input: An operand $c \in [0, Rp)$, a modulus p with $2^{N-1} < p < 2^N = R$, and a 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** $r \geq p$ **then**
4: $\quad r = r - p$
5: **end if**

Output: The residue $r = cR^{-1} \bmod p$.

The multi-precision version of Montgomery reduction, first described in [25] and then further improved in [37], has lower complexity than the standard Montgomery reduction and needs only $u^2 + u$ digit-multiplications. Alg. 4 specifies the multi-precision Montgomery reduction, which uses $R = 2^{uw}$ as Montgomery radix and replaces all operations by R in Alg. 3 with a loop performing operations by 2^w. As mentioned earlier, the prime p used in SIDH and SIKE has usually the form $p = f \cdot a^{e_A} b^{e_B} \pm 1$. If $a = 2$ and $w \leq e_A$, we have $p \bmod 2^w = \pm 1$. Therefore, the constant p' is equal to ∓ 1 and the multiplication by p' in line 3 can be omitted, which reduces the number of digit-multiplications to u^2. When w is the word size of the target processor architecture, we usually have $w \ll e_A$. This observation makes it possible to further reduce the complexity by refining the second multiplication to get rid of a loop-carried dependence, which was studied in full detail in [15] and [28]. More concretely, the complexity can be cut down 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 digit-multiplications decreases from 144 to only 84. Note that this method has been adopted by the designer of SIKE [17].

Algorithm 4: Multi-precision Montgomery reduction [37].

Input: An operand $c \in [0, Rp)$, a modulus p with $2^{N-1} < p < 2^N \leq 2^{uw} = R$, and a pre-computed constant $p' = (-p^{-1}) \bmod 2^w$.

1: **for** $i = 0$ to $u - 1$ **do**
2: $\quad t = ((c \bmod 2^w) \cdot p') \bmod 2^w$
3: $\quad c = (c + t \cdot p)/2^w$
4: **end for**
5: $r = c$
6: **if** $r \geq p$ **then**
7: $\quad r = r - p$
8: **end if**

Output: The residue $r = cR^{-1} \bmod p$.

The output of the Montgomery reduction is not $c \bmod p$ but $cR^{-1} \bmod p$, i.e. the result carries the factor R^{-1}. This makes necessary to convert the involved operands into the so-called Montgomery form at the beginning of an arithmetic operation, which is done by multiplying each operand by R. It has been shown that all prime-field operations can be carried out in Montgomery form in the same way as when using the normal representation. At the end of the arithmetic operation, the output has to be re-converted by dividing it by R.

2.3.2 Barrett Reduction Algorithm

The basic 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 $3u^2$ digit-multiplications.

The authors of [38] improved Barrett's model for the quotient q (in Step 1 of Alg. 5) by introducing two variable parameters σ and ρ. The quotient is written as:

$$q = \left\lfloor \frac{c}{p} \right\rfloor = \left\lfloor \frac{\frac{c}{2^{N+\rho}} \frac{2^{N+\sigma}}{p}}{2^{\sigma-\rho}} \right\rfloor \geq \left\lfloor \frac{\left\lfloor \frac{c}{2^{N+\rho}} \right\rfloor \left\lfloor \frac{2^{N+\sigma}}{p} \right\rfloor}{2^{\sigma-\rho}} \right\rfloor \qquad (3)$$

Assuming the estimation of the quotient is $\hat{q} = \left\lfloor \left\lfloor \frac{c}{2^{N+\rho}} \right\rfloor \left\lfloor \frac{2^{N+\sigma}}{p} \right\rfloor / 2^{\sigma-\rho} \right\rfloor$, the maximum error on the estimated quotient can be deduced according to the following formula from [39]:

Algorithm 5: Barrett reduction [27].

Input: An operand $c \in [0, 2^{2N+\gamma})$, a modulus p with $2^{N-1} < p < 2^N$, and a 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** $r \geq p$ **then**
4: $\quad r = r - p, q = q + 1$
5: **end if**

Output: The quotient $q = \lfloor c/p \rfloor$ and the remainder $r = c \bmod p$.

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

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

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

It can be easily shown that when $\sigma \geq N + \gamma + 1$ and $\rho \leq -2$, the estimation error is no larger than 1. The equality symbols in the previous equations were adopted to simplify the formulae. In this way, 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 Barrett reduction costs about $2u^2$ digit-multiplications, which is close to the original Montgomery reduction, but still inferior to the best Montgomery variant.

2.3.3 *Improved Barrett Reduction Algorithm for* 2^x3^y

Karmakar *et al.* refined in [26, Section 2.4] the original Barrett method (Alg. 5) to obtain an efficient division and reduction technique for moduli of the form 2^x3^y (where x and y are positive integers and $x \approx \log_2(3^y)$), which is a sub-operation of their special modular reduction algorithm for isogeny-based cryptography. Our new reduction algorithm adopts the approach from [26], as will be detailed in the next section.

The Improved Barrett Reduction (IBR) shown in Alg. 6 combines ideas from [26], [38], and [40]. More concretely, the splitting method in Step 1 is based on [26], the estimation method for the quotient in Step 2 is taken from [38], and the simplifications in Steps 3, 4, and 8 were originally described in [40]. The modulus is denoted as m (to distinguish it from the prime p) and the bit-length of the input c is assumed to be $2W + \gamma$. It can be seen that the first multiplication (Step 2) is a $(W + \gamma + 2) \times (W + \gamma + 1)$-bit multiplication and the second one (Step 3) a $(W_2+1) \times W_2$-bit multiplication. When the absolute value of γ is small and $W_1 \approx W_2 \approx W/2$, the IBR has a cost of 1.25 $W \times W$-bit multiplications. It should be noted that the IBR is not directly used as independent reduction algorithm for the SIKE protocol, but as a sub-operation to compute the quotient and remainder. Therefore, it can not be easily replaced by a Montgomery-based modular reduction algorithm. Table 1 shows a comparison of the complexity (in terms of the normalized number of $W + W$-bit additions and $W \times W$-bit multiplications) of different Barrett-based reduction algorithms, where γ is approximated to zero and $W_1 \approx W_2 \approx W/2$. We can see that the proposed IBR is the most efficient algorithm for moduli of the form 2^x3^y.

Algorithm 6: Improved Barrett reduction (IBR).

Input: An operand $c \in [0, 2^{2W+\gamma})$; a modulus $m = 2^x3^y$, where $W_1 = x$, $W_2 = \lceil \log_2(3^y) \rceil$, $W_1 + W_2 = W$, and $m' = 3^y$; and a 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: $\quad r = r - m', q = q + 1$
7: **end if**
8: $r = r \cdot 2^{W_1} + s$

Output: The quotient $q = \lfloor c/m \rfloor$ and the remainder $r = c \bmod m$.

TABLE 1
Comparison of the Normalized Number of $W + W$-bit Additions and $W \times W$-bit Multiplications of Different Barrett-Based Reduction Algorithms

Reference	[27]	[38]	[26]	[40]	IBR
Norm. $(W + W)$	3	3	2.5	1.5	1
Norm. $(W \times W)$	3	2	2	1.75	1.25

3 Proposed Data Representation for Supersingular Isogeny Based Cryptography

3.1 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, $n = \gcd(e_A + \alpha, e_B + \beta)$, $n \geq 1$, and $R = a^{\frac{e_A+\alpha}{n}} b^{\frac{e_B+\beta}{n}}$. We represent a field element $A \in \mathbb{F}_p$ using 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 $a_0 \in [0, R \pm 1]$, with the actual sign corresponding to the sign in the equation for p, i.e. Eq. (7). The correctness of this representation can be easily validated.

Our goal is to build a mapping that is defined for all elements of $\mathbb{F}_p$. When $p = f \cdot a^{e_A} b^{e_B} - 1$, the expression for A in Eq. (8) satisfies $0 \leq A \leq p$, whereas any element $A \in \mathbb{F}_p$ has to be in $[0, p-1]$, i.e. $0 \leq A < p$. Therefore, except for the largest value p, the values of the representation given by Eq. (8) can be exactly mapped to elements of $\mathbb{F}_p$. Similarly, when $p = f \cdot a^{e_A} b^{e_B} + 1$, the new representation can also express all elements of $\mathbb{F}_p$, but there is no one-to-one mapping in some cases. The proposed data representation can be regarded as a kind of redundant number system. We utilize the advantages of this representation for the reduction using the special form of the prime. All field-arithmetic operations needed for SIKE can be correctly performed using this new data representation. In the following, we will demonstrate how the proposed representation facilitates the implementation of arithmetic operations.

3.2 Deduction of A Low-Complexity Modular Reduction Algorithm

As mentioned before, the modular reduction is the most important of the basic arithmetic operations, and therefore we focus on it first. Let us consider the 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:

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

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. For the modulus $p = f' \cdot R^n \pm 1$, $C \bmod p$ is congruent to

$$C \equiv c_{n-1} R^{n-1} + \sum_{j=0}^{n-2} (c_j \mp \frac{c_{n+j}}{f'}) \cdot R^j \mod p. \tag{10}$$

The reduction operation also involves a conversion of the coefficients to bring them to the standard ranges as defined in Eq. (8). It should be noted that this equation is a general formula for any p of the form given by Eq. (7). For example, when $f' = 2$, $n = 2$, and $p = 2 \cdot R^2 - 1$, this result is equivalent to the equation derived by Karmakar *et al.* for their finite-field multiplication [26], which has been proven less efficient than the two multiplication techniques from [40]. As analyzed in [41], it seems that the larger n, the faster multiplication speeds can be achieved.

Note that the constant modular division $\frac{1}{f'} \bmod p$, whose result ranges in $(0, p)$, can be pre-computed. Clearly, when f' is restricted to an integer, the ideal case is $f' = 1$. The other cases would add a multiplication to each term and make the raw coefficient values large, which will increase the overall complexity of the reduction. When assuming $f' = 1$, we have $p = R^n \pm 1$. Since the modulus p must be prime, the minus sign is not possible for $n > 1$. However, the existing SIKE parameters adopt primes of the form $p = 2^{e_A} 3^{e_B} - 1$, where e_A and e_B are co-prime. When we directly use $f' = 1$ for the formula of Eq. (10), the modular reduction will not be very efficient since n can only be set to 1. In the following, we will demonstrate that this equation can become very simple for the primes of the SIKE when with $n > 1$.

Our goal is to make the modular division $\frac{1}{f'} \bmod p$ simple and fast. Obviously, the above approach is not efficient enough for the existing SIKE parameters. However, when we assume that f' is not an integer but a fraction with the numerator equal to 1, this problem can be easily solved. We found that, when small positive integers α and β are added to the exponents e_A and e_B (and f' is adapted accordingly), the exponents will not be co-prime and the greatest common divisor will usually be larger than 2 (specified by the parameter n in the fifth row of Table 3). In this case, we have $f' = 2^{-\alpha} 3^{-\beta}$ and f as defined in Eq. (7) equals 1. When going back to Eq. (10) with $p = 2^{-\alpha} 3^{-\beta} \cdot R^n - 1$, we get

$$C \equiv c_{n-1} R^{n-1} + \sum_{j=0}^{n-2} (c_j + c_{n+j} 2^{\alpha} 3^{\beta}) R^j \bmod p. \tag{11}$$

3.2.1 *Multiplication Part*

Equation (11) can be unfolded to

$$\begin{aligned} C \equiv &\sum_{i=0}^{n-1} a_i b_{n-i-1} \cdot R^{n-1} + \\ &\sum_{j=0}^{n-2} \Big(\sum_{i=0}^{j} a_i b_{j-i} + \sum_{i=j+1}^{n-1} a_i b_{j-i+n} \cdot 2^{\alpha} 3^{\beta} \Big) \cdot R^j \bmod p. \end{aligned} \tag{12}$$

Here, the process of computing the raw coefficients is defined as the integer multiplication operation. It should be noted that these multiply-accumulate terms look the same as those of the traditional unfolded multiplication, but their radices are different and the bit width of a coefficient in our algorithm equals $\lceil \frac{N}{n} \rceil$. Since the coefficients are independent from each other, our algorithm does not need to propagate carries from lower-order to higher-order coefficients. Additionally, as the coefficients are relatively small, a one-level Karatsuba-like optimization technique can be straightforwardly applied to the coefficient-product combinations of the form $a_i b_j + a_j b_i$ $(i \neq j)$ since

$$a_i b_j + a_j b_i = (a_i + a_j)(b_i + b_j) - a_i b_i - a_j b_j. \tag{13}$$

Note that the number of such combinations is $(n^2 - n)/2$. By using this method, the number of coefficient-products is reduced from n^2 to $(n^2 - n)/2 + n = n(n+1)/2$. Since α and β are generally very small (typically either 0 or 1), the constant multiplications in Eq. (12) boil down to simple additions and shift operations.

Microsoft's SIDH/SIKE library [17] implements the integer multiplication based on Comba's technique [42], which is a special scheduling method to effectively execute the

multiplication and accumulation of partial products. The analysis in [15] deems this technique more promising than the Karatsuba algorithm when the length of the operands is small to moderate. However, the number of multiply instructions to be executed is not reduced by Comba's scheduling method. In addition, a large number of carries have to be handled in the iterations. Thus, it is reasonable to assume that the proposed multiplication method can be faster than the previous one. Moreover, it should be noted that our multiplication method is naturally more suitable for massively-parallel implementations to achieve higher speed.

3.2.2 Low-Complexity Modular Reduction Algorithm

Regardless of which method is used to compute the raw coefficients of the product, the procedure of modular reduction for these coefficients is almost the same. Without loss of generality, we use a prime of the form $p = 2^{e_A}3^{e_B} - 1$ to explain the reduction. Suppose an integer $C = \sum_{j=0}^{n-1} c_j \cdot R^j$ with raw coefficients c_j computed via Eq. (12). All coefficients of the final result are required to be in the defined ranges, i.e. $c_j \in [0, R-1]$ for $0 \leq j < n-1$ and $c_{n-1} \in [0, f'R-1]$. To achieve this, the coefficients with index $0 \leq j < n-1$ have to be reduced by modulo R, whereby the remainders are kept as the corresponding new coefficients and the quotients are added to the adjacent higher-order coefficients. The $(n-1)$-th term requires a reduction modulo $f'R$, which can be performed according to the formula

$$\begin{aligned} & c_{n-1} \cdot R^{n-1} \mod p \\ \equiv & ((c_{n-1}R^{n-1} \bmod (f'R \cdot R^{n-1})) + \lfloor \frac{c_{n-1}R^{n-1}}{f'R^n} \rfloor) \bmod p \\ \equiv & ((c_{n-1} \bmod f'R) \cdot R^{n-1} + \lfloor \frac{c_{n-1}}{f'R} \rfloor) \bmod p. \end{aligned} \quad (14)$$

The obtained quotient has to be merged with the lowest term. Thereafter, several additions and subtractions are needed to adjust the final result. We execute $(n+1)$ IBR functions (presented in Alg. 6) to perform these 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 then the subtraction $e_j = d_j - R$, whereby R is replaced by $f'R$ for $j = n-1$. The updated quotient q_j is set as e_j's sign bit xored with 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 to the input c. Hence, the IBR functions can be optimized with respect to the corresponding largest input integers. Additionally, we adopt a "lazy" reduction technique for c_0, with a range of $[0, R]$, to reduce the required number of additions and subtractions. This trick is also used for the modular addition and subtraction presented in the next section.

The complexity of this new reduction algorithm can be evaluated via the number of multiply instructions. It should be noted that multiplications are only performed by the IBR functions. The IBR in Step 4 could be specifically optimized and become less costly than the other IBRs. For the sake simplicity, we use $((n-1)2^{\alpha}3^{\beta}+1)(R-1)^2$ as the largest input value for the IBR, whose 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 $2\lceil N/m \rceil$. A single IBR takes $1.25\ \lceil N/m \rceil \times \lceil N/m \rceil$-bit multiplications and the overall cost of the proposed reduction algorithm amounts to $1.25(n+1)\ \lceil N/m \rceil \times \lceil n/m \rceil$-bit multiplications.

When $\lceil N/n \rceil$ is equal to the word size w, n will be equal to u. However, $\lceil N/n \rceil$ is usually not exactly equal to w, while n still equals u. Taking SIKEp751 and $w = 64$ as example, we have $n = u = 12$, but $\lceil N/m \rceil = \lceil 751/12 \rceil = 63$ (this means about $1.25(n+1) = 16.25\ w \times w$-bit multiplications are needed). As explained in the previous section, the absolute value of λ is generally small. Consequently, the ratio of the best Montgomery reduction used in the SIKE library (which requires about $0.5n^2\ w \times w$-bit multiplications) to our reduction is approximately $0.4n^2/(n+1)$ in terms of multiplications. When $n > 3$, the proposed reduction algorithm may achieve better performance than the fastest Montgomery-based variant. The larger n, the faster speeds can be achieved, though this trend might weaken when targeting platforms with specific word sizes. Considering the small sizes of the IBR input operands, optimization methods like Karatsuba or Toom-Cook can be easily applied to the multiplication by utilizing the redundant representation. Finally, it should be noted that our reduction algorithm could be further accelerated by executing the IBR operations in parallel.

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. (12) 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]. When considering a specific software or hardware platform like FPGA, the IBRs in the G-IFFM can be computed in parallel to further improve the efficiency, similar to the strategy used in [26]. The final output can be adjusted by using a low-latency post-processing module.

The modular squaring in our implementation is separately designed since Eq. (13) equals $2a_ia_j$, which does not need 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 the additions as:

$$C = \sum_{j=0}^{n-1}(a_j + b_j) \cdot R^j = \sum_{j=0}^{n-1} c_j \cdot R^j. \tag{15}$$

Only n additions are cost without carries propagated in the neighbor orders. 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:

$$\begin{aligned} & c_{j+1} \cdot R^{j+1} + c_j \cdot R^j \\ &= (c_{j+1}+1) \cdot R^{j+1} + (c_j - R) \cdot R^j. \end{aligned} \tag{16}$$

Note that this step is similar to the ripple effect of standard carry propagation. If the condition is satisfied, one addition and one subtraction are required.

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

$$\begin{aligned} & c_{n-1} \cdot R^{n-1} + c_0 \bmod p \\ &\equiv c_{n-1} \cdot R^{n-1} - (f'R^n - 1) + c_0 \bmod p \\ &= (c_{n-1} - f'R) \cdot R^{n-1} + (1 + c_0) \bmod p. \end{aligned} \tag{17}$$

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.

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.

It seems that the serial propagation in reduction would limit the parallelism when considering a hardware platform like FPGA. In fact, we can still easily make it in parallel since the candidates for the final output are certain. For example, for the reduction, there are four candidates of the i-th coefficient, *i.e.*, c_i, $c_i + 1$, $c_i - R$, and $c_i - R + 1$. We can compute all of those candidates in parallel and select the right outputs by using the corresponding one-bit propagated quotients which consume very low latency for the propagation.

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 the subtractions as:

$$C = \sum_{j=0}^{n-1}(a_j - b_j) \cdot R^j = \sum_{j=0}^{n-1} c_j \cdot R^j. \tag{18}$$

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

$$\begin{aligned} & c_{j+1} \cdot R^{j+1} + c_j \cdot R^j \\ &= (c_{j+1}-1) \cdot R^{j+1} + (c_j + R) \cdot R^j. \end{aligned} \tag{19}$$

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

$$\begin{aligned} & c_{n-1} \cdot R^{n-1} + c_0 \bmod p \\ &\equiv c_{n-1} \cdot R^{n-1} + (f'R^n - 1) + c_0 \bmod p \\ &= (c_{n-1} + f'R) \cdot R^{n-1} + (c_0 - 1) \bmod p. \end{aligned} \tag{20}$$

The output coefficients c_j for $0 < j \leq n-1$ 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.

It can be seen that the reduction step is similar to the ripple effect of standard borrow propagation. Similarly, we can obtain a high degree of parallelism in hardware implementation by computing all the candidates in parallel and only propagating the one-bit borrows.

4.1.4 Modular Negation

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

$$\begin{aligned} & (-C) \bmod p \equiv (P - C) \bmod p \\ &= (f'R - c_{n-1} - 1) \cdot R^{n-1} + \sum_{j=0}^{n-2}(R - c_j - 1) \bmod p. \end{aligned} \tag{21}$$

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

4.1.5 Modular Inversion

According to the Fermat's little theorem [32], 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:

$$\frac{A}{2} \bmod p \equiv \begin{cases} \dfrac{A}{2}, & A \text{ is even}, \\ \dfrac{A+p}{2}, & A \text{ is odd}. \end{cases} \tag{22}$$

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 $0 < j \leq n-1$ is even, only a right shift is needed. Otherwise, we can compute it with the following decomposition:

$$\begin{aligned} \tfrac{a_j}{2} \cdot R^j &= (\frac{a_j - 1}{2} + \frac{1}{2}) \cdot R^j \\ &= (\frac{a_j - 1}{2}) \cdot R^j + \frac{R}{2} \cdot R^{j-1}. \end{aligned} \tag{23}$$

For the odd a_0, we have:

$$\frac{a_0}{2} = (\frac{a_0 - 1}{2}) + \frac{1}{2}. \tag{24}$$

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

$$\begin{aligned} \frac{1}{2} \bmod p &\equiv (\frac{p+1}{2}) \bmod p \\ = (\frac{f'R^n}{2}) \bmod p &= \frac{f'R}{2} \cdot R^{n-1} \bmod p. \end{aligned} \tag{25}$$

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

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

$$b_j = \begin{cases} \dfrac{a_j}{2}, & a_j \text{ is even}, \\ \dfrac{a_j - 1}{2}, & a_j \text{ is odd}, \end{cases} \quad \text{for } 0 \leq j \leq n-1; \tag{26}$$

$$c_j = \begin{cases} 0, & a_{j+1} \text{ is even}, \\ \dfrac{R}{2}, & a_{j+1} \text{ is odd}, \end{cases} \quad \text{for } 0 \leq j \leq n-2, \tag{27}$$

and

$$c_{n-1} = \begin{cases} 0, & a_0 \text{ is even}, \\ \dfrac{f'R}{2}, & a_0 \text{ is odd}. \end{cases} \tag{28}$$

Then, the division by two is computed as:

$$\sum_{j=0}^{n-1} \frac{a_j}{2} \cdot R^j \bmod p \equiv \sum_{j=0}^{n-1} (b_j + c_j) \cdot R^j \bmod p. \tag{29}$$

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. (22) 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:

$$\begin{aligned} A + B &= (A_0 + B_0) + (A_1 + B_1)i \bmod p, \\ A - B &= (A_0 - B_0) + (A_1 - B_1)i \bmod p, \\ A \times B &= (A_0B_0 - A_1B_1) + ((A_0 + A_1)(B_1 + B_0) \\ &\quad -A_0B_0 - A_1B_1)i \bmod p, \\ A^2 &= (A_0 + A_1)(A_0 - A_1) + (A_0A_1 + A_0A_1)i \bmod p, \\ A^{-1} &= A_0(A_0^2 + A_1^2)^{-1} + (-A_1(A_0^2 + A_1^2)^{-1})i \bmod p. \end{aligned} \tag{30}$$

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 2, 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 2
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. (30) 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:

$$\begin{aligned}&\sum_{j=0}^{n-1}(c_{0,j}-c_{1,j})\cdot R^j \bmod p \qquad (31)\\ \equiv&\sum_{j=0}^{n-1}(c_{0,j}-c_{1,j})\cdot R^j + xR(f'R^n-1) \bmod p\\ =&(c_{0,n-1}-c_{1,n-1}+xR(f'R-1))R^{n-1}+\\ &\sum_{j=0}^{n-2}(c_{0,j}-c_{1,j}+xR(R-1))\cdot R^j \bmod p,\end{aligned}$$

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: $\quad c_j \leftarrow A \bmod R$
3: $\quad A \leftarrow \lfloor \frac{A}{R} \rfloor$
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.

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: $\quad C \leftarrow C \cdot R + a_j$
4: **end for**
5: If $C = p$, set C to 0.
6: If $C = p+1$, set C to 1.

Output: The result $C \in \mathbb{F}_p = A \bmod p$.

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**[1] (in optimized portable C 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 3. 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^{64}, 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

1. It is implemented in optimized portable C code without using the GMP library. The original portable code using the GMP library, named "reference implementation" on the website of https://sike.org/, is not covered in this library.

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 algorithms in this paper. The overall results of the four parameters 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.

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 4. 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 5 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 multiplication instructions to implement it. Thus, 7 multiplication instructions are adopted for one IBR, and therefore, $13 \times 7 = 91$ 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 5) 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 $a_i b_j + a_j b_i$ ($i \neq j$) 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. (12), 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 5, 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 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 5, 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 by implementing on an x64 or ARM platform in assembly code. Nevertheless, it can still believe that the proposed modular addition/subtraction is able to 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

TABLE 3
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 4
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%

of modular addition and subtraction is not the bottleneck according to the statistics in Table 4.

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 2. 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 6. In all cases, we are able to 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 the other three parameters and tested the running time as shown in Tables 7, 8, and 9. It can be seen that our implementations obtain speedups of about 2.03x, 1.61x, and 1.48x over the previous works, respectively, 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 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, the proposed algorithms for SIKE are very suitable for high-parallel design 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. Recently, we have preliminarily completed the modular multiplication, addition, and subtraction on an FPGA and almost achieved the expected results. Our future work will mainly focus on those points to further bridge the gap.

Acknowledgement

The authors would like to thank the reviewers for their comments. This work is supported in parts by the National Natural Science Foundation of China (Grant No.61802180 and No.61774082), the Fundamental Research Funds for the Central Universities (Grant 021014380065), and the Key Research Plan of Jiangsu Province of China (Grant BE2019003-4).

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. 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.

[7] 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.

TABLE 5
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 6
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 7
Overall Timing Comparisons of the SIKEp610 Software Implementations. Timings Are Reported in Clock Cycles

Phase	[17]	Our work	AF
KeyGen	186,213,290	92,155,578	2.02
Encaps	168,786,347	172,206,327	2.03
Decaps	344,433,945	170,012,117	2.03
Total	872,717,924	430,954,042	2.03

TABLE 8
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

TABLE 9
Overall Timing Comparisons of the SIKEp434 Software Implementations. Timings Are Reported in Clock Cycles

Phase	[17]	Our work	AF
KeyGen	66,056,313	44,205,016	1.49
Encaps	106,052,319	71,809,157	1.48
Decaps	113,185,503	76,820,145	1.47
Total	285,294,135	192,834,318	1.48

[8] 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.

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

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

[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," 2020, 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.

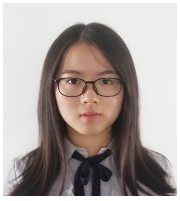
[24] B. Koziel, A.-B. Ackie, R. E. Khatib, R. Azarderakhsh, and M. Mozaffari-Kermani, "SIKE'd Up: Fast and secure hardware architectures for supersingular isogeny key encapsulation," Cryptology ePrint Archive, Report 2019/711, 2019, https://eprint.iacr.org/2019/711.

[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,"

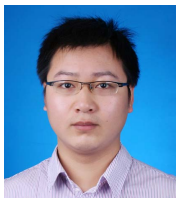
in *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 1986, pp. 311–323.
[28] J. Bos and S. Friedberger, "Arithmetic considerations for isogeny based cryptography," *IEEE Transactions on Computers*, pp. 1–1, 2018.
[29] D. Jao *et al.*, "Supersingular isogeny key encapsulation (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] F. Zhang, B. Yang, X. Dong, S. Guilley, Z. Liu, W. He, F. Zhang, and K. Ren, "Side-channel analysis and countermeasure design on arm-based quantum-resistant sike," *IEEE Transactions on Computers*, vol. 69, no. 11, pp. 1681–1693, 2020.
[32] E. W. Weisstein, "Fermat's little theorem," *From MathWorld–A Wolfram Web Resource*, 2004, https://mathworld.wolfram.com/FermatsLittleTheorem.html.
[33] 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.
[34] 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.
[35] A. Schönhage and V. Strassen, "Schnelle multiplikation grosser zahlen," *Computing*, vol. 7, no. 3-4, pp. 281–292, 1971.
[36] M. Fürer, "Faster integer multiplication," *SIAM Journal on Computing*, vol. 39, no. 3, pp. 979–1005, 2009.
[37] 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.
[38] 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.
[39] 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.
[40] 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.
[41] B. Wu, J. Tian, X. Hu, and Z. Wang, "A novel modular multiplier for isogeny-based post-quantum cryptography," in *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2020, pp. 334–339.
[42] P. G. Comba, "Exponentiation cryptosystems on the IBM PC," *IBM systems journal*, vol. 29, no. 4, pp. 526–538, 1990.



Jing Tian received her B.S. degree in microelectronics and Ph.D. degree in information and communication engineering from Nanjing University, Nanjing, China, in 2015 and 2020, respectively. She is now an associate researcher in Nanjing University. Her research interests include VLSI design for digital signal processing and cryptographic engineering.

Piaoyang Wang received his B.S. degree in electronic information science and technology from Nanjing University, Nanjing, China, in 2016, where he is currently pursuing the M.S. degree in Electronic and Communication Engineering. His research interests include VLSI design and cryptographic engineering.

Zhe Liu is a professor in the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China. He received the B.S. and M.S. degrees from Shandong University, China, in 2008 and 2011, respectively, and the Ph.D. degree from the Laboratory of Algorithmics, Cryptology and Security, University of Luxembourg, Luxembourg, in 2015. His research interests include security, privacy and cryptography solutions for the Internet of Things. He has co-authored over 100 research peer-reviewed journal and conference papers. He was a recipient of the prestigious FNR Outstanding Ph.D. Thesis Award in 2016, ACM CHINA SIGSAC Rising Star Award in 2017, as well as DAMO Academy Young Fellow in 2019. He has served as program committee member of more than 50 international conferences.



Jun Lin received the B.S. degree in physics and the M.S. degree in microelectronics from Nanjing University, Nanjing, China, in 2007 and 2010, respectively, and the Ph.D. degree in electrical engineering from the Lehigh University, Bethlehem, in 2015. From 2010 to 2011, he was an ASIC design engineer with AMD. During summer 2013, he was an intern with Qualcomm Research, Bridgewater, NJ. In June 2015, he joined the school of electronic science and engineering of Nanjing University, where he is an associate professor. He is a member of the Design and Implementation of Signal Processing Systems (DISPS) Technical Committee of the IEEE Signal Processing Society.

His current research interests include low-power high-speed VLSI design, specifically VLSI design for digital signal processing and cryptography. He was a co-recipient of the Merit Student Paper Award at the IEEE Asia Pacific Conference on Circuits and Systems in 2008. He was a recipient of the 2014 IEEE Circuits & Systems Society (CAS) student travel award.

Zhongfeng Wang received both BS and MS degrees from Tsinghua University. He obtained the Ph.D. degree from the University of Minnesota, Minneapolis, in 2000. He has been working for Nanjing University, China, as a Distinguished Professor since 2016. Previously he worked for Broadcom Corporation, California, from 2007 to 2016 as a leading VLSI architect. Before that, he worked for Oregon State University and National Semiconductor Corporation.

Dr. Wang is a world-recognized expert on Low-Power High-Speed VLSI Design for Signal Processing Systems. He has published over 200 technical papers with multiple best paper awards received from the IEEE technical societies, among which is the VLSI Transactions Best Paper Award of 2007. He has edited one book "VLSI" and held more than 20 U.S. and China patents. In the current record, he has had many papers ranking among top 25 most (annually) downloaded manuscripts in IEEE Trans. on VLSI Systems. In the past, he has served as Associate Editor for IEEE Trans. on CAS-I, T-CAS-II, and T-VLSI for many terms. He has also served as TPC member and various chairs for tens of international conferences. Moreover, he has contributed significantly to the industrial standards. So far, his technical proposals have been adopted by more than fifteen international networking standards. In 2015, he was elevated to the Fellow of IEEE for contributions to VLSI design and implementation of FEC coding. His current research interests are in the area of Optimized VLSI Design for Digital Communications and Deep Learning.

Johann Großschädl is a research (and development) specialist in the Laboratory of Algorithmics, Cryptology and Security (LACS) of the University of Luxembourg. His research interests lie at the intersection between theory and practice of modern cryptography, with a special focus on lightweight cryptography for the Internet of Things (IoT). Before joining the University of Luxembourg in 2009, he was a research associate in the Computer Science Department of the University of Bristol, United Kingdom. He has published more than 100 papers in international, peer-reviewed journals and conference proceedings, including more than 10 papers at the IACR Workshop on Cryptographic Hardware and Embedded Systems (CHES).