An Alternative Approach for SIDH Arithmetic

Cyril Bouvier(✉) and Laurent Imbert(✉)

LIRMM, Univ. Montpellier, CNRS, Montpellier, France
cyril.bouvier@lirmm.fr, laurent.imbert@lirmm.fr

Abstract. In this paper, we present new algorithms for the field arithmetic layers of supersingular isogeny Diffie-Hellman; one of the fifteen remaining candidates in the NIST post-quantum standardization process. Our approach uses a polynomial representation of the field elements together with mechanisms to keep the coefficients within bounds during the arithmetic operations. We present timings and comparisons for SIKEp503 and suggest a novel 736-bit prime that offers a 1.17× speedup compared to SIKEp751 for a similar level of security.

Keywords: Supersingular isogeny Diffie-Hellman · Polynomial Modular Number System · Efficient arithmetic

1 Introduction

Driven by recent advances in quantum computing and the potential threat on public-key cryptography [5,17], the National Institute of Standards and Technology (NIST) launched a post-quantum cryptography standardization process. Sixty-nine public-key encryption and signature algorithms entered the competition in 2017. After over a year of evaluation, NIST revealed a list of 26 candidates selected for a second round of evaluation. At the end of July 2020, a list of seven third-round finalists (4 encryption/KEMs, 3 digital signature schemes) together with eight alternate candidates (5 encryption/KEMs, 3 digital signature schemes) was made public.

One of the encryption/KEMs 3rd-round alternate candidate is based on the Supersingular Isogeny Diffie-Hellman (SIDH) scheme proposed in 2011 by Jao and De Feo [12]. Its security relies on the hardness of computing a smooth-degree isogeny between two supersingular elliptic curves. The resulting NIST proposal, called SIKE for Supersingular Isogeny Key encapsulation [11], includes various optimizations from recent works such as [7] and [8]. SIKE is the only candidate based on isogenies between elliptic curves. A noteworthy advantage of SIKE over the other candidates is its very small public key sizes – the smallest of all encryption and KEMs schemes – as well as very small cyphertexts. However, as pointed out in [1]: "The main drawback to SIKE is that its performance (measured in clock cycles) is roughly an order of magnitude worse than many of its

J. A. Garay (Ed.): PKC 2021, LNCS 12710, pp. 27–44, 2021.
https://doi.org/10.1007/978-3-030-75245-3_2

competitors. Much work has been done to optimize implementations, including the compressed-key version, and it is hoped that such optimizations continue."

This is exactly the purpose of the present work dedicated to the arithmetic of SIDH. The theoretical foundations of isogeny-based cryptography are beyond the scope of this paper. We refer the newcomer to this field of research to [6,9,10,18] to only cite a few.

SIDH requires intensive computations in a quadratic extension of the finite field $\mathbb{F}_p$, where p is a prime of the form $p = c \cdot p_A^{e_A} \cdot p_B^{e_B} \pm 1$ for some primes p_A, p_B and some small c. The smallest value between $p_A^{e_A}$ and $p_B^{e_B}$ dictates the security level of the protocol. The primes p_A and p_B are thus chosen so that $p_A^{e_A}$ and $p_B^{e_B}$ are roughly the same size; preferably $p_A^{e_A} \approx p_B^{e_B} \approx \sqrt{p}$.

SIKE targets four security levels. The NIST proposal contains two algorithms called SIKE.PKE for public-key encryption and SIKE.KEM for key encapsulation mechanism. Both are implemented with four sets of parameters denoted SIKEpxxx, where xxx $\in \{434, 503, 610, 751\}$ corresponds to the bitlength of p. For all of these primes, $c = 1$, $p_A = 2$, $p_B = 3$ and $p \equiv 3 \mod 4$. The quadratic extension $\mathbb{F}_{p^2}$ can thus be represented as $\mathbb{F}_p(i)$ with $i^2 = -1$. The arithmetic layers implemented in SIKE are already highly optimized. The field level notably makes use of a very efficient Montgomery reduction that benefits from the special form of p (see [4] for more details).

In this paper, we propose a new, efficient way to perform the arithmetic in $\mathbb{F}_{p^2}$ for these special SIDH primes. Our arithmetic relies on the concept of Polynomial Modular Number Systems (PMNS) proposed by Bajard, Imbert and Plantard [2] as an alternative approach for performing the arithmetic modulo N. In a PMNS, the elements are represented as polynomials of bounded degree, and the basic arithmetic operations are carried out using polynomial arithmetic. In Sect. 2, we extend the initial definition of PMNS to any finite field $\mathbb{F}_{p^k}$ and we present generic algorithms for the conversions from and into PMNS, and for the basic arithmetic operations. Ideally, we want the polynomial coefficients to be as small as possible. The main difficulty, only partially solved in [2], is to perform the arithmetic operations while keeping these coefficients small. In Sect. 3, we present a Montgomery-like coefficient reduction algorithm first suggested in [16] for prime fields $\mathbb{F}_p$, that can be used to solve this problem. In Algorithm 1, we give an extended generic version that works for any finite field $\mathbb{F}_{p^k}$. The principal contribution of this work is a special case of PMNS perfectly suited to the arithmetic of SIDH. We present optimized arithmetic algorithms and some implementation details in Sect. 4. Finally, we illustrate the efficiency of our approach with some experimental results and some comparisons with the SIKE parameters in Sect. 5. In particular, we suggest a new prime $p736$ which outperforms SIKEp751 by approximately 17% for a similar level of security. Our code is available at https://gitlab.inria.fr/ciao/pmns-for-sidh.

During the development of this work, a preprint posted on the IACR eprint archive [19] suggested a "new data representation", used to improve the arithmetic of SIDH. In reality, this "new" representation is a PMNS representation, which the authors did not seem to be aware of. Their coefficient reduction strategy is inspired by a modified Barrett algorithm from [13]. Unfortunately, their

implementation is an order of magnitude slower than both the optimized version of SIKE and the present work.

2 PMNS for Finite Fields

The Polynomial Modular Number System (PMNS) [2] was introduced to perform arithmetic in rings $\mathbb{Z}/N\mathbb{Z}$, with N being any positive integer, using integer polynomial arithmetic. It was used, in particular, to perform arithmetic in prime fields. In this section, we will extend the definition of PMNS representation to include all finite fields.

Definition 1 (PMNS basis and γ-representation). *Let p be a prime, k a positive integer and $\mathbb{F}_{p^k}$ be the finite field with p^k elements. Moreover, let n be a positive integer and E a degree-n polynomial such that E has a root $\gamma \in \mathbb{F}_{p^k}$ of algebraic degree k. Let Γ denotes the minimal polynomial of γ.*

The tuple $\mathcal{B} = (p, k, n, \Gamma, E, \gamma)$ is called a PMNS basis for the field $\mathbb{F}_{p^k}$.

A polynomial $V \in \mathbb{Z}[X]$ of degree $< n$ such that $V(\gamma) = v$, with $v \in \mathbb{F}_{p^k}$, is called a γ-representation of v in the PMNS basis $\mathcal{B}$.

Note that, by definition of γ, the polynomial Γ has degree exactly k; and since Γ is the minimal polynomial of γ, we also have $k \leq n$. Thus, for all v in $\mathbb{F}_{p^k}$, the expansion of v in the basis $(1, \gamma, \ldots, \gamma^{k-1})$ of $\mathbb{F}_{p^k}$ is a trivial γ-representation for v. Therefore, every elements of $\mathbb{F}_{p^k}$ admits a γ-representation. However, an element of $\mathbb{F}_{p^k}$ may have multiple γ-representations. We say that two integer polynomials U and V are equivalent if they represent the same element in $\mathbb{F}_{p^k}$, i.e. if $U(\gamma) = V(\gamma)$.

Example 2. The parameters $p = 19$, $k = 1$, $n = 3$, $\Gamma = X - 7$, $E = X^3 - 1$, $\gamma = 7$ define a PMNS basis for the prime field $\mathbb{F}_{19}$. It is easy to verify that $\Gamma(\gamma) \equiv E(\gamma) \equiv 0 \pmod{19}$. In Table 1, we list all the γ-representations of the elements of $\mathbb{F}_{19}$ with coefficients in $\{-1, 0, 1\}$.

Example 3. The parameters $p = 5$, $k = 2$, $n = 4$, $\Gamma = X^2 - 2$, $E = X^4 + 1$, $\gamma = \sqrt{2}$ define a PMNS basis for the field $\mathbb{F}_{5^2}$. It is easy to verify that $\Gamma(\gamma) \equiv E(\gamma) = 0$ in $\mathbb{F}_{5^2}$. Considering this PMNS basis with coefficients bounded by 2 in absolute value, we can see that, for example, $3\sqrt{2}$ admits two γ-representations ($-X^3$ and $X^3 + X$) and $3 + \sqrt{2}$ admits four γ-representations ($X^3 - X^2 - X$, $-X^2 + X$, $X^3 + X^2 - X + 1$, $X^2 + X + 1$).

Although γ-representations always exist, in practice, we are interested in polynomials with small coefficients.

Definition 4 (reduced representations). *Let $\mathcal{B}$ be a PMNS basis and let ρ be a positive integer. A γ-representation V of $v \in \mathbb{F}_{p^k}$ is said to be ρ-reduced if all the coefficients of the polynomial V are less than ρ in absolute value, i.e. $|v_i| < \rho$, for $0 \leq i < n$. (In the rest of this article, we may simply use the term reduced when there is no ambiguity on ρ.)*

Table 1. γ-representations of the elements of $\mathbb{F}_{19}$ with coefficients in $\{-1, 0, 1\}$ in the PMNS basis $\mathcal{B} = (19, 1, 3, X - 7, X^3 - 1, 7)$.

$v \in \mathbb{F}_{19}$	γ-representations	$v \in \mathbb{F}_{19}$	γ-representations
0	$-X^2 - X - 1$, 0, $X^2 + X + 1$	10	$X^2 - 1$
1	$-X^2 - X$, 1	11	$-X - 1$, X^2
2	$-X^2 - X + 1$	12	$-X$, $X^2 + 1$
3	$X^2 - X - 1$	13	$-X + 1$
4	$X^2 - X$	14	$-X^2 + X - 1$
5	$X^2 - X + 1$	15	$-X^2 + X$
6	$X - 1$	16	$-X^2 + X + 1$
7	$-X^2 - 1, X$	17	$X^2 + X - 1$
8	$-X^2$, $X + 1$	18	-1, $X^2 + X$
9	$-X^2 + 1$		

In practice, we will work with ρ-reduced γ-representations, with well chosen values of ρ. If ρ is too small, it may happen that some elements of $\mathbb{F}_{p^k}$ have no ρ-reduced γ-representation. A lower bound on values ρ for which all elements of $\mathbb{F}_{p^k}$ have a ρ-reduced γ-representation can be computed by considering a particular lattice associated to the PMNS basis. For a PMNS basis $\mathcal{B}$, let $\mathcal{L}_{\mathcal{B}}$ denote the lattice over $\mathbb{Z}^n$ generated by the set of vectors corresponding to integer polynomials $Z \in \mathbb{Z}[X]$ of degree at most $n - 1$ such that $Z(\gamma) = 0$ in $\mathbb{F}_{p^k}$. An elementary basis of $\mathcal{L}_{\mathcal{B}}$ is given by the n polynomials $p, pX, \ldots, pX^{k-1}$, $X^k - \gamma^k$, $X^{k+1} - \gamma^k X, \ldots, X^{n-1} - \gamma^k X^{n-1-k}$, or equivalently by the following $n \times n$, integer row-matrix

$$\mathcal{L}_{\mathcal{B}} = \begin{pmatrix} p & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & \ddots & & & & & \vdots \\ \vdots & & \ddots & & & & \vdots \\ 0 & \cdots & 0 & p & 0 & 0 & \cdots 0 \\ -\gamma^k & 0 & \cdots & 0 & 1 & 0 & \cdots 0 \\ 0 & \ddots & & & & \ddots & \vdots \\ \vdots & & \ddots & & & & \ddots 0 \\ 0 & \cdots & 0 & -\gamma^k & 0 & \cdots & 0 \; 1 \end{pmatrix}. \tag{1}$$

Let A be the row-matrix of any basis of $\mathcal{L}_{\mathcal{B}}$ and let $||A||_1$ be its 1-norm, defined as $||A||_1 = \max\{\sum_{i=0}^{n} |a_{i,j}|\ ,\ 0 \leq j < n\}$. Then, using [3, Theorem 2.2] we know that if $\rho > \frac{1}{2}||A||_1$, then there exist at least one ρ-reduced γ-representation for every element of $\mathbb{F}_{p^k}$. This result means that any lattice reduction algorithm (assuming n is not too large) such as LLL or BKZ can be used to determine a lower bound on the values of ρ that are valid to define a notion of reduction for a PMNS basis.

In the next sections, we present generic algorithms for the conversions from and into PMNS and for the basic arithmetic operations. As we shall see, the main and only stumbling block is to ensure that the output results have reduced coefficients.

2.1 Conversions and Basic Arithmetic Operations

From PMNS: Given a γ-representation $V \in \mathbb{Z}[X]$, computing the corresponding element in $\mathbb{F}_{p^k}$ is just a matter of evaluating V at γ, with arithmetic operations carried out in $\mathbb{F}_{p^k}$.

To PMNS: Given $v \in \mathbb{F}_{p^k}$, computing a γ-representation for v is equivalent to writing v in the basis $(1, \gamma, ..., \gamma^{k-1})$. However, in general, the obtained representation is not reduced.

Addition/Subtraction: A γ-representation of the sum $u+v$ (resp. difference $u-v$) is computed by adding (resp. subtracting) the corresponding polynomials U and V. Again, the resulting polynomial may not be reduced.

Multiplication: A γ-representation of the product uv can be computed in two steps. We first compute $W = U \times V$ so that $W(\gamma) = U(\gamma)V(\gamma) = uv$ in $\mathbb{F}_{p^k}$. Since $\deg W \leq 2n-2$, we reduce the degree by computing $W' = W \bmod E$. We have $W' = W - QE$ for some polynomial Q, with $\deg W' < \deg E = n$. Since γ is a root of E, $W'(\gamma) = W(\gamma)$. Thus, W' is a γ-representation of uv. However, as for the addition, this representation is not reduced.

Note that the polynomial E can be freely chosen. In practice, we will often choose $E = X^n - e$, with $e \in \mathbb{Z}$, as it allows for faster algorithms. It is also easier to compute useful bounds in this case. For example, if U and V are both ρ-reduced γ-representations, then it is not difficult to see that the coefficients of $W' = UV \bmod (X^n - e)$ are bounded by $n|e|\rho^2$ in absolute value.

2.2 Coefficient Reduction

Let $\mathcal{B} = (p, n, k, \Gamma, E, \gamma)$ be a PMNS basis and $\mathcal{L}_\mathcal{B}$ be the associated lattice given in (1). According to [3], we know that if $\rho > \frac{1}{2}||A||_1$ for some basis A of $\mathcal{L}_\mathcal{B}$, then all the elements of $\mathbb{F}_{p^k}$ admit a ρ-reduced γ-representation. Computing a ρ-reduced γ-representation for $v \in \mathbb{F}_{p^k}$ from V, an equivalent non-reduced one, amounts to reducing the vector corresponding to the polynomial V modulo the lattice $\mathcal{L}_\mathcal{B}$. We thus need a way to efficiently find a lattice vector that is close enough to V. (In general we do not need to compute the closest vector to V.)

In [2], Bajard et al. suggested to perform this operation through lookup tables. A lattice vector close to V is retrieved from a precomputed table, using the most significant bits of the coefficients of V. The retrieved vector is then subtracted from V to obtain a reduced representation. The size of the precomputed table depends on the number of bits that need to be reduced. This approach may be appropriate for a very few bits, but becomes unpractical as the number of bits to reduce grows. In order to manage larger reductions, the authors of [2] present various strategy to reduce the size of the required tables at the cost of some extra arithmetic operations.

The next section presents an alternative approach that do not require any table. First suggested in [16] for prime fields $\mathbb{F}_p$, it naturally extends to any finite field $\mathbb{F}_{p^k}$. It is inspired by Montgomery multiplication/reduction [15]. Therefore, it is only stable in the so-called Montgomery representation, and can only be used after a multiplication. In Sect. 4.2, we detail our reduction strategy after an addition in the case of special PMNS basis for SIDH.

3 PMNS Coefficient Reduction à la Montgomery

Given a γ-representation C of some field element $c \in \mathbb{F}_{p^k}$, Algorithm 1 below computes a γ-representation of $c/2^\omega$ whose coefficients are approximately ω bits smaller than those of c. The value 2^ω plays the same role as the Montgomery constant. In practice ω is chosen according to the size of ρ; the smaller integer multiple of the word-size larger than ρ is a common choice. The algorithm also needs a γ-representation of zero, denoted M in Algorithm 1. Any representation of zero that is invertible in $(\mathbb{Z}/2^\omega\mathbb{Z})[X]/(E)$ is an acceptable choice for M.

Algorithm 1. (Generic) Montgomery coefficient reduction

Input: C a γ-representation of $c \in \mathbb{F}_{p^k}$, M a γ-representation of zero and M' the inverse of $-M$ in $(\mathbb{Z}/2^\omega\mathbb{Z})[X]/(E)$.
Output: R a γ-representation of $c/2^\omega$
1: $Q \leftarrow CM' \bmod E \bmod 2^\omega$
2: $R \leftarrow (C + QM \bmod E)/2^\omega$
3: **return** R

In order to prove the correctness of Algorithm 1, simply observe that in the ring $(\mathbb{Z}/2^\omega\mathbb{Z})[X]/(E)$, the polynomial $C + QM$ is zero, so the last division is exact. Moreover, since $M(\gamma) = 0$ in $\mathbb{F}_{p^k}$, we have $(C + QM)(\gamma) = C(\gamma)$ in $\mathbb{F}_{p^k}$ so that R is a γ-representation of $c/2^\omega$.

In general, it is difficult to derive useful bounds on the size of the output coefficients. However, in the case $E = X^n - e$, with $e \in \mathbb{Z}$, we can show that the size of the output coefficients are approximately ω bits smaller than those of the input. If we assume that M is ρ-reduced and that the coefficients of C are less than 2^t in absolute value, with t a positive integer, then the coefficients of

$C+QM$ mod E are less than $2^t+2^\omega n|e|\rho$ in absolute value. Thus, the coefficients of the output are less than $2^{t-\omega}+n|e|\rho$ in absolute value.

In the rest of this paper, we concentrate our attention on a special class of PMNS basis well suited to SIDH. In particular, we shall explain how this generic algorithm can be used to efficiently reduce the coefficients after a multiplication of two reduced γ-representations.

4 PMNS for SIDH

In SIDH, arithmetic operations are performed in $\mathbb{F}_{p^2}$ for a prime p of the form $c \cdot p_A^{e_A} \cdot p_B^{e_B} \pm 1$, where p_A, p_B are two distinct primes. For efficiency reasons, a common choice is to opt for $p_A = 2, p_B = 3, c = 1$. In this section, we will show that PMNS basis are very well-suited to SIDH. We will describe our *special PMNS basis for SIDH* in a slightly more general setting.

Definition 5. *Let p be an odd prime of the form $p = |\gamma^n/e - 1|$, where γ is an element of $\overline{\mathbb{F}_p}$ of algebraic order $k > 0$ such that γ^k is an integer; n is a positive integer; and e is an integer divisor of γ^k. Note that since p is an integer, we must have $k|n$. A PMNS basis for $\mathbb{F}_{p^k}$ of the form $(p, k, n, X^k - \gamma^k, X^n - e, \gamma)$ is called a* special PMNS basis for SIDH.

Proposition 6. *If $\mathcal{B} = (p, k, n, X^k - \gamma^k, X^n - e, \gamma)$ is a special PMNS basis for SIDH, then the polynomial $M = (\gamma^k/e)X^{n-k} - 1$ is a γ-representation of zero in $\mathcal{B}$.*

Proof. Simply observe that $M(\gamma) = \pm p$. □

In addition to being a representation of zero, the polynomial M is also very sparse, with exactly two nonzero coefficients. Moreover, since $\gamma^k \approx p^{k/n}$, these coefficients are "small". As will be explained in details in Sect. 4.3, these properties, which come from the special form of p, are essential for the efficiency of the Montgomery reduction algorithm.

Remark 7. Given $\mathcal{B}$ a special PMNS basis for SIDH, a small basis of the lattice $\mathcal{L}_\mathcal{B}$ associated to $\mathcal{B}$ (as defined in (1)) is given by the n vectors corresponding to the n polynomials M, XM, $\cdots$, $X^{k-1}M$, Γ, $X\Gamma$, $\cdots$, $X^{n-k-1}\Gamma$. The 1-norm of the columns of the corresponding matrix is either $|\gamma^k| + 1$ or $|\gamma^k/e| + 1$. Thus, the 1-norm of the matrix is $|\gamma^k| + 1$ for all $e \in \mathbb{Z}$. Using [3, Th. 2.2], this implies that $(|\gamma^k| + 1)/2$ is a lower bound for ρ. In other terms, it guarantees that if $\rho > (|\gamma^k| + 1)/2$, then any $v \in \mathbb{F}_{p^k}$ admits a ρ-reduced γ-representation in $\mathcal{B}$.

Example 8. Let $p = 2^{250}3^{159} - 1$ be the prime called SIKEp503 in the SIKE [11] submission to the NIST post-quantum standardization process. Special PMNS basis for SIDH make it possible to represent the elements of $\mathbb{F}_p$, but also those of $\mathbb{F}_{p^2}$ directly.

- A special PMNS basis for SIDH for the prime field $\mathbb{F}_p$, may be obtained by writing $p = (2^{25}3^{16})^{10}/3 - 1$, i.e. $k = 1$, $n = 10$, $e = 3$, and $\gamma = 2^{25}3^{16}$. We have $\Gamma = X - 2^{25}3^{16}$ and $E = X^{10} - 3$. Any value ρ greater than the 50-bit integer $\gamma/2 = 2^{24}3^{16}$, can be used to define reduced representations. In particular, the polynomial $M = 2^{25}3^{15}X^9 - 1$ is a (ρ-reduced) γ-representation of zero. In this case, the extension field $\mathbb{F}_{p^2}$ may be defined as $\mathbb{F}_p(i)$ where $i^2 = -1$.
- Alternatively, a special PMNS basis for SIDH can be built directly for the quadratic extension $\mathbb{F}_{p^2}$ by writing $p = (2^{62}3^{40}\sqrt{-2})^4/3 - 1$, i.e. $k = 2$, $n = 4$, $e = 3$, and $\gamma = 2^{62}3^{40}\sqrt{-2}$. We have $\Gamma = X^2 + 2^{125}3^{80}$ and $E = X^4 - 3$. Any value ρ greater than the 252-bit integer $\gamma^2/2 = 2^{125}3^{80}$, can be used to define reduced representations. In particular, the polynomial $M = -2^{125}3^{79}X^2 - 1$ is a (ρ-reduced) γ-representation of zero.

4.1 Conversions from and into the Representation

Converting an element of $\mathbb{F}_{p^k}$ to a special PMNS basis for SIDH is done in two steps. First, we write the element in the basis $(1, \gamma, \cdots, \gamma^{k-1})$. This can be done with integer coefficients in $[0, p[$. Then, we need to write each integer coefficient in radix $|\gamma^k|$ (recall than γ^k is an integer) with integer digits in $\left[-|\gamma^k|/2, |\gamma^k|/2\right[$. Each coefficient has at most n/k digits. Hence, the output is a polynomial of degree less than n whose coefficients are less than or equal to $|\gamma^k|/2$ in absolute value. In other words, it is ρ-reduced for every possible ρ-value, according to Remark 7.

The reverse conversion, from a special PMNS basis for SIDH to an element of $\mathbb{F}_{p^k}$, is performed by evaluating the polynomial at γ.

4.2 Coefficients Reduction for Additions and Subtractions

The ability to quickly reduce the coefficients after the various arithmetic operations is crucial for the overall efficiency our approach.

Let us first consider the reduction of the coefficients after additions and subtractions that only increase the coefficients' size by a few bits.

The proposed reduction algorithm is presented in Algorithm 2. For the algorithm to work, we require ρ to be larger than $|\gamma^k|$, i.e. twice the lower bound given by remark 7. The operations in line 6 correspond to an euclidean division by ρ, with signed remainder. Since ρ can be freely chosen, it is judicious to pick a power of 2 that makes this operation very fast and easy to implement.

Theorem 9. *Let $\mathcal{B}$ be a special PMNS basis for SIDH, with $\rho > |\gamma^k|$, and let t a fixed positive integer. If $\max|R| + |e|\max|Q| < \rho/2$, then Algorithm 2 is correct.*

Proof. At the end of the algorithm, we have

$$V \equiv U + (X^k - \gamma^k)\sum_{i=0}^{n-1} c_i X^i \pmod{E}$$

which means that $V(\gamma) = U(\gamma)$ in $\mathbb{F}_{p^k}$. Hence U and V are equivalent in $\mathcal{B}$.

Algorithm 2. Coefficient reduction

Input: a special PMNS basis for SIDH $\mathcal{B} = (p, k, n, X^k - \gamma^k, X^n - e, \gamma)$; a positive integer t; and an element U of $\mathcal{B}$ of degree at most $n-1$ with $|u_i| < t\rho$ for $0 \leq i < n$.
Output: a reduced element V of $\mathcal{B}$ of degree at most $n-1$ with $|v_i| < \rho$ for $0 \leq i < n$, equivalent to U.

$Q[j] \leftarrow \lfloor j\rho/\gamma^k \rceil$ for $0 \leq j \leq t$ ▷ precomputations
$R[j] \leftarrow j\rho - \gamma^k Q[j]$ for $0 \leq j \leq t$ ▷ precomputations

function CoeffReduc(U)
  $c_j \leftarrow 0$ for $-k \leq j < 0$
  for $i = 0$ to $n-1$ **do**
    Write u_i as $s_i \times u_{i,h} \times \rho + u_{i,\ell}$ with $s_i = \pm 1$, $u_{i,h} \geq 0$ and $|u_{i,\ell}| \leq \rho/2$
    $v_i \leftarrow u_{i,\ell} + s_i \times R[u_{i,h}] + c_{i-k}$
    $c_i \leftarrow s_i \times Q[u_{i,h}]$
  $v_j \leftarrow v_j + e \times c_{n-k+j}$ for $0 \leq j < k$
  return V

Now, for $0 \leq i < k$, we have $c_{i-k} = 0$, thus

$$\begin{aligned} v_i &= u_{i,\ell} + s_i \times R[u_{i,j}] + c_{i-k} + e \times c_{n-k+j} \\ &\leq \frac{\rho}{2} + \max|R| + e \times \max|C| < \rho \end{aligned}$$

And for $k \leq i < n$, $v_i = u_{i,\ell} + s_i \times R[u_{i,j}] + c_{i-k}$, and the same bound is achieved with the same arguments. □

In practice we will use this algorithm with $t = 2$ or $t = 4$, the precomputed tables will be of length 3 or 5. The elements of R are stored on as many bits as $|\gamma^k|$ and, if ρ is close to $|\gamma^k|$, the elements of Q are approximately t-bit long. For larger values of t, the precomputed tables would be too large, so, for example, Algorithm 2 cannot be used to perform the reduction after a multiplication.

Example 10. With the special PMNS basis for SIDH for SIKEp503 from Example 8: $\mathcal{B} = (\text{SIKEp503}, 1, 10, X - 2^{25}3^{16}, X^{10} - 3, 2^{25}3^{16})$, ρ will be chosen as the smallest power of 2 larger than γ, i.e. $\rho = 2^{51}$. The condition of Theorem 9 is satisfied for $t = 2$, $t = 4$, $t = 128$, and even for $t = 2^{20}$, but this later value is too large to be practical.

4.3 Montgomery Multiplication

Finally, we need to adapt the new Montgomery reduction algorithm described in Algorithm 1 in the case of a special PMNS basis for SIDH and show that it can be used after a multiplication to obtain a reduced representation.

Algorithm 1, requires a representation of zero that is invertible in the quotient ring $(\mathbb{Z}/2^\omega\mathbb{Z})[X]/(E)$. In the case of a special PMNS basis for SIDH, we exhibited the good candidate M for a representation of zero in Proposition 6. The next

theorem demonstrates that M can indeed be used in the Montgomery reduction algorithm and also gives a explicit formula for its inverse.

Theorem 11. *Let $\mathcal{B} = (p, k, n, \Gamma, E, \gamma)$ be a special PMNS basis for SIDH and let $M = (\gamma^k/e)X^{n-k} - 1$ be the representative of zero in $\mathcal{B}$ introduced in Proposition 6. Let ω be a positive integer such that $\gamma^{n+k}/e^2 \equiv 0 \pmod{2^\omega}$, and let β be the largest integer such that $\gamma^{\beta k}/e \not\equiv 0 \pmod{2^\omega}$ (the assumption on ω implies that $\beta \leq n/k$). Then, the inverse of the polynomial $-M$ in $(\mathbb{Z}/2^\omega\mathbb{Z})[X]/(E)$ is given by:*

$$M' = 1 + \sum_{i=1}^{\beta} \frac{\gamma^{ik}}{e} X^{n-ik}$$

Proof. Let us show that $M \times M' = -1$ in $(\mathbb{Z}/2^\omega\mathbb{Z})[X]/(E)$.

$$\begin{aligned} M \times M' &= \left(\frac{\gamma^k}{e}X^{n-k} - 1\right) \times \left(1 + \sum_{i=1}^{\beta} \frac{\gamma^{ik}}{e} X^{n-ik}\right) \\ &= \frac{\gamma^k}{e}X^{n-k} + \sum_{i=1}^{\beta} \frac{\gamma^{(i+1)k}}{e^2} X^{2n-(i+1)k} - \sum_{i=1}^{\beta} \frac{\gamma^{ik}}{e} X^{n-ik} - 1 \\ &= \sum_{i=2}^{\beta+1} \frac{\gamma^{ik}}{e^2} X^{2n-ik} - \sum_{i=2}^{\beta} \frac{\gamma^{ik}}{e} X^{n-ik} - 1. \end{aligned}$$

Since $E = X^n - e$, we have $X^n \equiv e \pmod{E}$. Therefore

$$\begin{aligned} M \times M' &\equiv \sum_{i=2}^{\beta} \frac{\gamma^{ik}}{e} X^{n-ik} + \frac{\gamma^{(\beta+1)k}}{e^2} X^{2n-(\beta+1)k} - \sum_{i=2}^{\beta} \frac{\gamma^{ik}}{e} X^{n-ik} - 1 \pmod{E} \\ &\equiv \frac{\gamma^{(\beta+1)k}}{e^2} X^{2n-(\beta+1)k} - 1 \pmod{E}. \end{aligned}$$

If $2n - (\beta+1)k \geq n$ (i.e. $\beta k < n$), then, after reduction by $E = X^n - e$, we have

$$M \times M' = \frac{\gamma^{(\beta+1)k}}{e} X^{n-(\beta+1)k} - 1$$

which is equal to -1 in $(\mathbb{Z}/2^\omega\mathbb{Z})[X]/(E)$ by definition of β. Finally, if $\beta k = n$, we conclude using the fact that $\gamma^{n+k}/e^2 \equiv 0 \bmod 2^\omega$. □

Note that for SIDH[1], the assumption $\gamma^{n+k}/e^2 \equiv 0 \pmod{2^\omega}$ is easy to satisfy as γ is divisible by a large power of 2. In practice, the choice of ω and the fact that 2^w is almost half the size of γ will often imply that β, which corresponds to the number of non-constant monomials of M' is 2.

Example 12. Let us consider again the special PMNS basis for SIDH for SIKEp503 from Example 8: $\mathcal{B} = (\text{SIKEp503}, 1, 10, X - 2^{25}3^{16}, X^{10} - 3, 2^{25}3^{16})$.

[1] With $p_A = 2$.

On a 64-bit architecture, we may want to use $\omega = 64$. The condition $\gamma^{n+k}/e^2 = 2^{275}3^{174} \equiv 0 \pmod{2^{64}}$ is easily satisfied. Since $\gamma^2 = 2^{50}3^{32} \not\equiv 0 \pmod{2^{64}}$ and $\gamma^3 = 2^{75}3^{48} \equiv 0 \pmod{2^{64}}$, we have $\beta = 2$. Thus

$$M' = \frac{\gamma}{e}X^9 + \frac{\gamma^2}{e}X^8 + 1.$$

Theorem 13. *Using the same notations as in Theorem 11. Let ρ be a bound on the coefficient size such that $\rho > |\gamma^k|$. Let C be an element of $\mathcal{B}$ with coefficients bounded by $2^\omega\rho$ in absolute value. Then, Algorithm 1 applied to C will return an element of $\mathcal{B}$ with coefficients bounded by 2ρ in absolute value.*

Proof. Considering that the coefficients of Q are bounded by 2^ω and taking into account the special form of M, the product $QM \mod E$ has its coefficients bounded by $2^\omega|\gamma^k| + 2^\omega$ in absolute value. So the coefficients of the polynomial returned by Algorithm 1 are bounded by

$$\frac{|c_i| + 2^\omega|\gamma^k| + 2^\omega}{2^\omega} < \frac{2^\omega\rho + 2^\omega|\gamma^k| + 2^\omega}{2^\omega} = \rho + |\gamma^k| + 1 \leq 2\rho.$$

□

4.4 Implementation Details of Montgomery Reduction

Theorem 14. *Let $(p, k, n, X^k - \gamma^k, X^n - e, \gamma)$ be a special PMNS basis for SIDH. Let ω, β, M and M' be defined as in Theorem 11. Let $C = \sum_{i=0}^{n-1} c_i X^i$ be a integer polynomial and $Q = \sum_{i=0}^{n-1} q_i X^i$ be the product $CM' \bmod E \bmod 2^\omega$ from line 1 of Algorithm 1. The coefficients of Q are given by:*

$$q_i = \begin{cases} c_i + \sum_{j=1}^{\beta} \frac{\gamma^{jk}}{e} c_{i+jk-n} \mod 2^\omega & \text{if } n-k \leq i < n \\ c_i + \gamma^k q_{i+k} \mod 2^\omega & \text{if } 0 \leq i < n-k \end{cases}$$

Moreover,

$$C + QM = \sum_{i=0}^{n-k-1} (c_i + \gamma^k q_{i+k} - q_i)X^i + \sum_{i=n-k}^{n-1} (c_i + \frac{\gamma^k}{e} q_{i-n+k} - q_i)X^i \mod E$$

Proof.

$$
\begin{aligned}
CM' &= (\sum_{i=0}^{n-1} c_i X^i)(1 + \sum_{j=1}^{\beta} \frac{\gamma^{jk}}{e} X^{n-jk}) = \sum_{i=0}^{n-1} c_i X^i + \sum_{j=1}^{\beta} \sum_{i=0}^{n-1} \frac{\gamma^{jk}}{e} c_i X^{n-jk+i} \\
&\equiv \sum_{i=0}^{n-1} c_i X^i + \sum_{j=1}^{\beta} \left(\sum_{i=0}^{jk-1} \frac{\gamma^{jk}}{e} c_i X^{n-jk+i} + \sum_{i=jk}^{n-1} \gamma^{jk} c_i X^{i-jk} \right) \pmod{E} \\
&\equiv \sum_{i=0}^{n-1} c_i X^i + \sum_{j=1}^{\beta} \left(\sum_{i=n-jk}^{n-1} \frac{\gamma^{jk}}{e} c_{i-n+jk} X^i + \sum_{i=0}^{n-1-jk} \gamma^{jk} c_{jk+i} X^i \right) \\
&\equiv \sum_{i=0}^{n-1} \left(c_i + \sum_{j=1}^{\min(\lfloor \frac{n-i-1}{k} \rfloor, \beta)} \gamma^{jk} c_{i+jk} + \sum_{j=\lfloor \frac{n-i-1}{k} \rfloor + 1}^{\beta} \frac{\gamma^{jk}}{e} c_{i+jk-n} \right) X^i
\end{aligned}
$$

The formula for q_i in the case $n-k \leq i < n$ is proven from the above formula by noticing that in this case $\lfloor (n-i-1)/k \rfloor = 0$.

To prove the formula in the case $0 \leq i < n-k$, we need the two following facts: first that $\lfloor (n-(i+k)-1)/k \rfloor = \lfloor (n-i-1)/k \rfloor - 1$ and then that multiplying q_{i+k} by γ^k is equivalent to shifting the indexes in the sum. To conclude, it remains to use the fact that $\gamma^{(\beta+1)k}/e \equiv 0 \pmod{2^\omega}$.

The formula from $C + QM$ comes from a straightforward computation of the product QM mod E using the particular form of $M = (\gamma^k/e)X^{n-k} - 1$. □

This theorem proves that Algorithm 3 is a correct adaption of Algorithm 1 in the case of a special PMNS basis for SIDH. It is more efficient as it performs only a linear number of multiplications: β multiplications modulo 2^ω per iteration in the first loop and 1 full multiplication per iteration for the second and third loops. The remaining operations are additions and shifts.

Algorithm 3. Montgomery reduction for special PMNS basis for SIDH

Input: as in Algorithm 1
Output: as in Algorithm 1

1: **for** $j = n-1$ down to $n-k$ **do**
2: $\quad q_j \leftarrow c_j + \sum_{i=1}^{\beta} (\gamma^{ik}/e) c_{j+ik-n} \mod 2^\omega$
3: **for** $j = n-k-1$ down to 0 **do**
4: $\quad t_j \leftarrow c_j + \gamma^k q_{j+k}$
5: $\quad (r_j, q_j) \leftarrow (\lfloor t_j / 2^\omega \rfloor, t_j \bmod 2^\omega)$
6: **for** $j = n-1$ down to $n-k$ **do**
7: $\quad t_j \leftarrow c_j + (\gamma^k/e) q_{j-n+k}$
8: $\quad r_j \leftarrow \lfloor t_j / 2^\omega \rfloor$
9: **return** R

Algorithm 3 compute the q_j's starting from q_{n-1} down to q_0. It first computes $q_{n-1}, \ldots, q_{n-k}$ using the first case of the formula from Theorem 14. Then for j from $n-k-1$ to 0, it computes the full product $\gamma^k q_{j+k} + c_j$, uses it to compute the j-th coefficient of $(C+QM)/2^\omega$ and takes it modulo 2^ω to obtain q_j. Note that, while computing the j-th coefficient of $(C+QM)/2^\omega$, we do not need to subtract q_j before dividing by 2^ω as the two operands are equal modulo 2^ω.

5 Results

In this section we validate the efficiency of our novel arithmetic through two special PMNS basis for SIDH. We compare our implementation with the code available on the Github repository PQCrypto-SIDH [14][2], more precisely with the uncompressed, optimized x64 implementation. We implemented the PMNS arithmetic in assembly and did not make any changes to the code handling the elliptic operations nor the SIDH and SIKE functions. For our new prime $p736$, we generated the necessary sets of parameters as well as new optimized strategies for the isogeny computations using the specifications from [11].

The operations of conversions into and from PMNS are only computed once before and after communications between the parties as the protocol specifies the format of the exchange data. For the coordinates of the public generators points of the SIKE protocol, we replaced their value in the source code by a reduced PMNS representation.

As pointed out in the introduction, an approach similar to ours was independently proposed in [19]. We did not include their timings in our comparisons since their implementation is about ten times slower than both the optimized implementation of SIKE and the present work.

Note that the arithmetic of SIKE implemented in [14] is already highly optimized. For the quadratic extension field $\mathbb{F}_p(i)$, it uses a Karatsuba-like algorithm for the multiplication but only performs two modular reductions (instead of three). For the arithmetic in $\mathbb{F}_p$, it benefits from the special form of p. Indeed, for $p = 2^{e_2}3^{e_3} - 1$, the Montgomery modular reduction may be greatly optimized. First, because the multiplication by p may be replaced by a multiplication by $p+1 = 2^{e_2}3^{e_3}$, which itself reduces to a multiplication by 3^{e_3} plus some shifts, followed by a subtraction. Second, because the inverse of p modulo the Montgomery constant R (chosen as the smallest multiple of the word-size larger than p) is equal to -1 modulo 2^w for $w = 32, 64$, which further reduces the overall number of word multiplications. More details are given in [4].

All comparisons were performed on a desktop computer with a 3.2 GHz Intel Core i7-8700 (Coffee Lake) processor with Hyper-Threading and TurboBoost disabled, running Ubuntu 18.04.5 and gcc 6.5.0. The compilation options used are identical to the ones in the Makefile provided by [14].

The code is available at https://gitlab.inria.fr/ciao/pmns-for-sidh. The current version of this article corresponds to commit `fc666429`.

[2] We use commit `4eb51ae0` (few commits after tag version 3.3).

5.1 SIKEp503

As a proof of concept, and for comparisons and compatibility purposes, we generated two different special PMNS basis for SIKEp503 $= 2^{250}3^{159} - 1$.

Our first special PMNS basis uses polynomials of degree 9 with coefficients on a single 64-bit word. We used the following parameters:

$$k = 1 \qquad n = 10 \qquad \gamma = 2^{25}3^{16}$$
$$\Gamma = X - \gamma \qquad E = X^{10} - 3 \qquad \rho = 2^{56}$$

In this case, we implemented Algorithm 3 with $\omega = 64$.

We also consider a second special PMNS basis with degree-2 polynomials and coefficients on three 64-bit words; hence a total of nine 64-bit words.

$$k = 1 \qquad n = 3 \qquad \gamma = 2^{84}3^{53}$$
$$\Gamma = X - \gamma \qquad E = X^{3} - 4 \qquad \rho = 2^{170}$$

In this case, we implemented Algorithm 3 with $\omega = 192$.

In Table 2, we give the number of clock cycles for the main arithmetic operations in our two PMNS basis denoted 10×1 and 3×3 respectively and we compare them to the corresponding operations from the optimized reference implementation of SIKE [14]. We split the cost of a $\mathbb{F}_p$ multiplication into the actual multiple-precision or polynomial product, followed by the Montgomery reduction. In the PMNS cases, the product corresponds to the polynomial multiplication modulo E, and the Montgomery reduction corresponds to the coefficient reduction algorithm (Algorithm 3) presented in Sect. 4.4.

Table 2. SIKEp503: cost of field operations in number of clock cycles

	[14]	This work 10×1	Speedup	This work 3×3	Speedup
$\mathbb{F}_p$ Addition	19	22	0.86	19	1.00
$\mathbb{F}_p$ Multiplication	143	131	1.09	139	1.03
Product	72	113	0.64	108	0.67
Montgomery reduction	55	23	2.39	33	1.67
$\mathbb{F}_{p^2}$ Addition	35	41	0.85	35	1.00
$\mathbb{F}_{p^2}$ Multiplication	358	446	0.80	423	0.85
$\mathbb{F}_{p^2}$ Square	308	318	0.97	300	1.03

Despite a much faster Montgomery reduction, and a slightly faster multiplication in $\mathbb{F}_p$, our multiplication in $\mathbb{F}_{p^2}$ remains slower than that of SIKEp503. This is mainly due to the fact that our field elements are represented on two extra words for the 10×1 variant and one extra word for the 3×3 variant.

Therefore, our product requires more word-size multiplications. And since the multiplication in $\mathbb{F}_{p^2}$ uses 3 products but only two reductions, the overall performance of the original SIKE implementation is better. As can be seen in Table 3, these results translate immediately to the key exchange and key encapsulation schemes.

5.2 SIKEp751 Versus Our p736

The prime SIKEp751 $= 2^{372}3^{239} - 1$ in the NIST submission was proposed for the largest security level. For an equivalent level of security, we propose a new 736-bit prime, called p_{736} in the following. In order to build a special PMNS basis for the finite field $\mathbb{F}_p$ for $p = p_{736} = 2^{361}3^{236} - 1$, we used the following parameters:

$$k = 1 \qquad n = 4 \qquad \gamma = 2^{91}3^{59}$$
$$\Gamma = X - \gamma \qquad E = X^4 - 8 \qquad \rho = 2^{186}$$

Table 3. SIKEp503: cost of SIDH and SIKE operations in number of clock cycles

	[14]	This work 10×1	Speedup	This work 3×3	Speedup
Key exchange					
Alice's key generation	5322813	6584635	0.81	6188158	0.86
Bob's key generation	5927772	7254013	0.82	6781400	0.87
Alice's shared key computation	4361263	5324520	0.82	4988913	0.87
Bob's shared key computation	5036883	6182503	0.81	5695130	0.88
Kem					
Key generation	5946153	7297933	0.81	6784183	0.88
Encapsulation	9726092	11925016	0.82	11174206	0.87
Decapsulation	10359163	12688804	0.82	11877351	0.87

In this case, we implemented Algorithm 3 with $\omega = 192$. This PMNS basis uses degree-3 polynomials and coefficients on three 64-bit words; hence a total of twelve 64-bit words, the same number of words used for the arithmetic of SIKEp751.

For this prime, we choose to use 2^{360} instead of 2^{361} for Alice's subgroup order, in order to be able to use only 4-isogeny in the key computation.

As for SIKEp503 above, we compared the field arithmetic operations for both SIKEp751 and our p_{736}. The results presented in Table 4 exhibit a 10% speedup for the multiplication and 29% for the square in $\mathbb{F}_{p^2}$. And as a consequence, a 1.17× speedup factor for the key exchange and key encapsulation schemes (see Table 5).

Table 4. p_{736} and SIKEp751: cost of field operations in number of clock cycles

	[14] SIKEp751	This work p_{736}	Speedup
$\mathbb{F}_p$ Addition	29	22	1.32
$\mathbb{F}_p$ Multiplication	274	198	1.38
Product	140	162	0.86
Montgomery reduction	106	39	2.72
$\mathbb{F}_{p^2}$ Addition	54	40	1.35
$\mathbb{F}_{p^2}$ Multiplication	693	631	1.10
$\mathbb{F}_{p^2}$ Square	559	435	1.29

Table 5. p_{736} and SIKEp751: cost of SIDH and SIKE operations in number of clock cycles

	[14] SIKEp751	This work p_{736}	Speedup
Key exchange			
Alice's key generation	15836023	13625479	1.16
Bob's key generation	17945236	15609183	1.15
Alice's shared key computation	13040542	11140436	1.17
Bob's shared key computation	15368807	13326150	1.15
Kem			
Key generation	17975299	15609821	1.15
Encapsulation	28849145	24735564	1.17
Decapsulation	31317267	26879415	1.17

6 Conclusions

We presented new algorithms to perform the arithmetic for primes used in the context of SIDH. These algorithms uses a polynomial representation of the field elements based on the existing Polynomial Modular Number System. We proposed new techniques to control the size of the coefficients of those polynomial representations which are particularly effective for the primes used in the context of SIDH.

We show that our new approach is competitive with the optimized implementation accompanying the SIKE submission to the NIST post-quantum standardization process. For the largest security level, we proposed a new prime that offers a 1.17× speedup compared to SIKEp751.

As seen for SIKEp503, different PMNS basis can be constructed for a given prime. Playing with the polynomial degree and the coefficient's sizes offers many optimization options for implementing the field arithmetic operations that

should be further investigated. Moreover, as explained in Example 8, PMNS can handle elements of $\mathbb{F}_{p^2}$ directly. This nice feature could make it possible to consider primes for SIDH that are congruent to 1 (mod 4). But, for SIDH primes of the form $p = c \cdot 2^{e_2} \cdot 3^{e_3} + 1$, square roots of integers of the form $\pm 2^a 3^b$ always exist in $\mathbb{F}_p$, which prevents us from using a γ^k of this form in Definition 5 to build a special PMNS basis for SIDH directly for $\mathbb{F}_{p^2}$ in this case. However, we believe that extra improvements and more primes of interest for SIDH are at hand.

Acknowledgements. In the submitted version of this work, the Montgomery-like coefficient reduction algorithm was presented as an original contribution. We are most grateful to thank Pascal Véron who pointed out Christophe Nègre and Thomas Plantard's paper [16] that actually contains the original algorithm.

This work was supported by the French *Agence Nationale de la Recherche, projet CIAO (ANR-19-CE48-0008).*

References

1. Alagic, G., et al.: Status report on the second round of the NIST post-quantum cryptography standardization process. Technical report NISTIR 8309, National Institute of Standards and Technology, U.S. Department of Commerce, July 2020. https://doi.org/10.6028/NIST.IR.8309
2. Bajard, J.-C., Imbert, L., Plantard, T.: Arithmetic operations in the polynomial modular number system. In: Proceedings of the 17th IEEE Symposium on Computer Arithmetic, ARITH17, pp. 206–213. IEEE Computer Society (2005). https://doi.org/10.1109/ARITH.2005.11
3. Bajard, J.-C., Marrez, J., Plantard, T., Véron, P.: On polynomial modular number systems over $\mathbb{Z}/p\mathbb{Z}$ (2020). https://hal.sorbonne-universite.fr/hal-02883341
4. Bos, J.W., Friedberger, S.: Faster modular arithmetic for isogeny-based crypto on embedded devices. J. Cryptogr. Eng. **10**(2), 97–109 (2020). https://doi.org/10.1007/s13389-019-00214-6
5. Buchanan, W., Woodward, A.: Will quantum computers be the end of public key encryption? J. Cyber Secur. Technol. **1**(1), 1–22 (2017). https://doi.org/10.1080/23742917.2016.1226650
6. Costello, C.: Supersingular isogeny key exchange for beginners. In: Paterson, K.G., Stebila, D. (eds.) SAC 2019. LNCS, vol. 11959, pp. 21–50. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-38471-5_2
7. Costello, C., Longa, P., Naehrig, M.: Efficient algorithms for supersingular isogeny Diffie-Hellman. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9814, pp. 572–601. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_21
8. Faz-Hernández, A., López-Hernández, J.C., Ochoa-Jiménez, E., Rodríguez-Henríquez, F.: A faster software implementation of the supersingular isogeny diffie-hellman key exchange protocol. IEEE Trans. Comput. **67**(11), 1622–1636 (2018). https://doi.org/10.1109/TC.2017.2771535
9. De Feo, L.: Mathematics of isogeny based cryptography (2017)
10. Galbraith, S.D., Vercauteren, F.: Computational problems in supersingular elliptic curve isogenies. Cryptology ePrint Archive, Report 2017/774 (2017). https://eprint.iacr.org/2017/774

11. Jao, D., et al.: SIKE - supersingular isogeny key encapsulation. Submission to the NIST Post-Quantum Standardization project (2017), sike.org
12. Jao, D., De Feo, L.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: Yang, B.-Y. (ed.) PQCrypto 2011. LNCS, vol. 7071, pp. 19–34. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25405-5_2
13. Karmakar, A., Roy, S.S., Vercauteren, F., Verbauwhede, I.: Efficient finite field multiplication for isogeny based post quantum cryptography. In: Duquesne, S., Petkova-Nikova, S. (eds.) WAIFI 2016. LNCS, vol. 10064, pp. 193–207. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-55227-9_14
14. Longa, P., et al.: PQCrypto-SIDH (2020). https://github.com/microsoft/PQCrypto-SIDH
15. Montgomery, P.L.: Modular multiplication without trial division. Math. Comput. **44**(170), 519–521 (1985)
16. Negre, C., Plantard, T.: Efficient modular arithmetic in adapted modular number system using Lagrange representation. In: Mu, Y., Susilo, W., Seberry, J. (eds.) ACISP 2008. LNCS, vol. 5107, pp. 463–477. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70500-0_34
17. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th Annual Symposium on Foundations of Computer Science, pp. 124–134 (1994). https://doi.org/10.1109/sfcs.1994.365700
18. Smith, B.: Pre- and post-quantum Diffie-Hellman from groups, actions, and isogenies. In: Budaghyan, L., Rodríguez-Henríquez, F. (eds.) WAIFI 2018. LNCS, vol. 11321, pp. 3–40. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-05153-2_1
19. Tian, J., Wang, P., Liu, Z., Lin, J., Wang, Z., Großschadl, J.: Faster Software Implementation of the SIKE Protocol Based on A New Data Representation. Cryptology ePrint Archive, Report 2020/660 (2020). https://eprint.iacr.org/2020/660