Supersingular Isogeny-Based Cryptography: Implementation Aspects and Parameter Selection

Patrick Longa

Microsoft Research, USA
plonga@microsoft.com

Abstract. These notes, written for the Isogeny-based Cryptography School (2021), cover implementation aspects of supersingular isogeny-based protocols, with special focus on SIDH and SIKE. The techniques and algorithms presented here are, for example, used in the SIDH library [13]. The document also includes a discussion of the cryptanalysis and parameter selection for SIKE.

1 Introduction

The computational layers that make up supersingular isogeny key-exchange protocols can be visualized in Fig. 1. In these notes, we will focus on the algorithmic and implementation aspects that correspond to the layers below the protocol level. We start by describing the curve and isogeny arithmetic.

2 Curve and isogeny arithmetic

For efficiency and compactness purposes, supersingular isogeny-based protocols are most commonly instantiated using Montgomery curves, which we describe next closely following [5].

2.1 Montgomery curves and their arithmetic

A Montgomery curve [14] over a finite field $\mathbb{F}_q$ with $\mathrm{char}(\mathbb{F}_q) \neq 2$ is defined by the equation

$$E_{(a,b)} : by^2 = x^3 + ax^2 + x,$$

where $(a, b) \in \mathbb{A}^2(\mathbb{F}_q), a^2 \neq 4$ and $b \neq 0$. For practical instantiations of supersingular isogeny-based schemes, it is more efficient to work *fully* in the projective space for both the curve points and the curve coefficients, as proposed by Costello et al. [5]. In this case, the Montgomery curve equation can be re-written as

$$E_{(A:B:C)} : By^2 = Cx^3 + Ax^2 + Cx,$$

where $(A \colon B \colon C) \in \mathbb{P}^2(\mathbb{F}_q)$ with $C \in \bar{\mathbb{F}}_q^{\times}$ is such that $a = A/C$ and $b = B/C$. The notation $(X \colon Y \colon Z) \in \mathbb{P}^2(K)$ with $Z \neq 0$ represents all the points $(x, y) = (X/Z, Y/Z)$ in $\mathbb{A}^2(\mathbb{F}_q)$, and the point at infinity is $\mathcal{O} = (0 : 1 : 0)$.

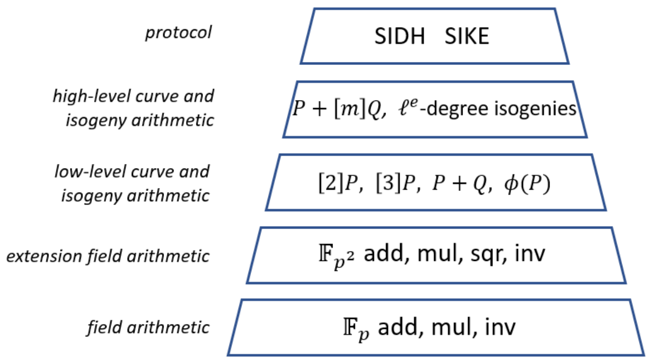
protocol
SIDH SIKE
high-level curve and isogeny arithmetic
$P + [m]Q$, ℓ^e-degree isogenies
low-level curve and isogeny arithmetic
$[2]P$, $[3]P$, $P + Q$, $\phi(P)$
extension field arithmetic
$\mathbb{F}_{p^2}$ add, mul, sqr, inv
field arithmetic
$\mathbb{F}_p$ add, mul, inv

Fig. 1: Computational layers of supersingular isogeny key-exchange protocols.

Kummer varieties and points in $\mathbb{P}^1$. To enable fast and compact implementations one can work with the Kummer varieties of Montgomery curves defined by $E_{(A:B:C)}/\langle\pm 1\rangle$. In this case, points are mapped to $\mathbb{P}^1$ using

$$\chi : \ E_{(A:B:C)} \setminus \{\mathcal{O}\} \to \mathbb{P}^1, \ (X : Y : Z) \mapsto (X : Z), \ \mathcal{O} \mapsto \{1, 0\}.$$

We write $\chi(P) = (X : Z)$ to represent a point in the Kummer variety.

Let P, Q be points in $E_{(A:B:C)} \setminus \{\mathcal{O}\}$. We define the doubling function `xDBL` : $(\chi(P), A, C) \mapsto \chi([2]P)$, the differential addition function `xADD` : $(\chi(P), \chi(Q), \chi(Q-P)) \mapsto \chi(Q+P)$, and the function `xDBLADD` : $(\chi(P), \chi(Q), \chi(Q-P), A, C) \mapsto (\chi([2]P), \chi(Q-P))$ that merges the first two[1]. These functions are all that is needed for the realization of the so-called Montgomery ladder, which computes the function `LADDER` : $(\chi(P), A, C, m) \mapsto \chi([m]P)$. In supersingular isogeny-based protocols, one also makes use of two additional functions: the Montgomery tripling function `xTPL` : $(\chi(P), a) \mapsto \chi([3]P)$, and the "three-point ladder" function `LADDER_3_pt` : $(\chi(P), \chi(Q), \chi(Q-P), A, C, m) \mapsto \chi(P + [m]Q)$, which is applied to the computation of the kernels [7].

Note that the `xADD` function for $E_{(A:\,B:\,C)}$ is identical to the original formula for $E_{(a,b)}$ due to Montgomery [14]. Other functions on $E_{(a,b)}$ that involve the Montgomery coefficient a can be easily modified to work on $E_{(A:\,B:\,C)}$ by substituting $a = A/C$ and then carrying the denominator C through to the projective output to avoid the associated inversion [5]. This optimization takes advantage of the fact that an elliptic curve and its non-trivial quadratic twist under the quotient by $\langle\pm 1\rangle$ share the same Kummer variety and, hence, their arithmetic is independent of the Montgomery coefficient b (or B).

[1] We note that the differential addition fails for $Q - P \in \{\mathcal{O}, (0, 0)\}$.

Algorithm 1 Montgomery "three-point ladder" function `LADDER_3_pt` : $(\chi(P), \chi(Q), \chi(Q-P), A, C, m) \mapsto \chi(P + [m]Q)$.

Require: $m = (m_{t-1}, \ldots, m_0)_2 \in \mathbb{Z}, (x_P, x_Q, x_{Q-P})$, and $(A : 1)$.
Ensure: $\chi(P + [m]Q) = (X_{P+[m]Q} : Z_{P+[m]Q})$

```
1: $\chi(R_0) = (x_Q : 1)$, $\chi(R_1) = (x_P : 1)$, $\chi(R_2) = (x_{Q-P} : 1)$
2: for $i = 0, \ldots, t-1$ do
3:     if $m_i = 0$ then
4:         $(\chi(R_0), \chi(R_2)) \leftarrow$ xDBLADD$(\chi(R_0), \chi(R_2), \chi(R_1), A, 1)$
5:     else
6:         $(\chi(R_0), \chi(R_1)) \leftarrow$ xDBLADD$(\chi(R_0), \chi(R_1), \chi(R_2), A, 1)$
7: return $\chi(R_1)$
```

2.2 Computation of the large-, smooth-degree isogenies

Let E be a starting, public curve[2], and P, Q be two points of order r lying on E. For the computation of a large-degree ℓ^e-isogeny, we need to compute a kernel point with the form $[k]P + [l]Q$, for $k, l \in \mathbb{Z}_r$, and its corresponding isogeny $\phi : E \to E/\langle [k]P + [l]Q \rangle$. Typically, the final goal is to compute the image curve $E/\langle [k]P + [l]Q \rangle$, but in some cases we also need to evaluate ϕ at some points of E.

Kernel computation $P + [m]Q$. De Feo et al. [7] observed that any generator of $\langle [k]P + [l]Q \rangle$ suffices to compute the required isogenies. If we assume k is invertible modulo r, then the kernel point simplifies to $P + [lk^{-1}]Q$, which is equivalent to directly using $P + [m]Q$ for some $m \in \mathbb{Z}_r$.

Computing $P + [m]Q$ can be done efficiently using *one* `xDBLADD` function (i.e., one doubling and one differential addition) per each bit of the scalar m, as shown in Algorithm 1. This algorithm, proposed by Faz-Hernández et al. [9], improves over the three-point ladder algorithm originally proposed to implement SIDH which required one doubling and two differential additions per scalar bit [7]. Note that Algorithm 1 assumes that both P and Q vary at each call[3]. When the points are fixed[4], it is possible to do better by storing the intermediate values corresponding to R_0, i.e., the points $[2^i]Q$. If we store the full t multiples, the cost of computing $P + [m]Q$ reduces to one differential addition per bit of the scalar, injecting a roughly 2x speedup (see [9, Alg. 3]).

Isogeny computation and evaluation. The large-degree ℓ^e-isogeny computation can be visualized as traversing a tree, from top to bottom, doing point multiplications by ℓ and ℓ-isogeny computations which are guided by a so-called

[2] E.g., in SIDH, E can be the fixed, starting curve of the protocol, or the curve that is part of Alice's or Bob's public key.

[3] E.g., this fits the case in the SIDH protocol in which the points P and Q are passed to the other party as part of a public key.

[4] E.g., this fits the case in the key generation stage of SIDH.

optimal strategy. This optimal strategy is derived by using the relative cost of a point multiplication by ℓ and an ℓ-isogeny evaluation. Since the topic of optimal strategies has already been covered in the school (Week 5: "Advanced SIDH Protocols"), we skip it here and refer the reader to [7, §4.2.2] and [3, §1.3.8] for more details.

Recall that $\ell \in \{2,3\}$ for key-exchange protocols like SIDH and SIKE. The atomic isogeny operations that are required for these schemes are degree-4 and degree-3 isogenies (it turns out that computing degree-4 isogenies is much faster than using degree-2 isogenies). We describe the formulas for the 4-isogenies below.

Projective degree-4 isogenies. Let $\chi(P) = (X_4 : Z_4) \in \mathbb{P}^1$ be such that P has order 4 in $E_{(A:C)}$. Let $E'_{(A':C')} = E_{(A:C)}/\langle P\rangle$, $\phi : E_{(A:C)} \to E'_{(A':C')}$, and $Q \in E_a \setminus \ker(\phi)$. A first formula is derived to compute the isogenous curve $E_{(A' : C')}$ from $E_{(A: C)}$ and $(X_4 : Z_4)$:

$$(A' : C') = \big(2(2X_4^4 - Z_4^4) \,:\, Z_4^4\big). \tag{1}$$

An improved version of formula (1) replaces $(A' : C')$ with the coefficients $(A' + 2C' : 4C')$ throughout the full isogeny computation. In this case, we simply have

$$(A' + 2C' : 4C') = \big(4X_4^4 \,:\, 4Z_4^4\big), \tag{2}$$

This optimization is used, for example, in the SIDH and SIKE implementations in the SIDH library [13].

A second formula is used to evaluate the isogeny by computing $\chi(\phi(Q)) = (X' : Z') \in \mathbb{P}^1$ from the additional input $\chi(Q) = (X : Z) \in \mathbb{P}^1$:

$$\begin{aligned}(X' : Z') = \big(c_2(X-Z) + c_1(X+Z)\big)^2 \big(c_0(X+Z)(X-Z) + {}&\\ \big(c_2(X-Z) + c_1(X+Z)\big)^2\big) :&\\ \big(c_2(X-Z) - c_1(X+Z)\big)^2 \big(\big(c_2(X-Z) + c_1(X+Z)\big)^2&\\ - c_0(X+Z)(X-Z)\big),&\end{aligned} \tag{3}$$

where $c = [\,c_0, c_1, c_2\,] = [\,Z_4^2, X_4 - Z_4, X_4 + Z_4\,]$ are values that are stored and re-used from eq. (2), taking advantage of the fact that each 4-isogeny is typically evaluated at multiple points.

The computation of eq. (2) and of the three values in c above costs $4\mathbf{S} + 5\mathbf{A}$[5], and on input of c and $\chi(Q) = (X : Z)$, the computation of eq. (3) costs $6\mathbf{M} + 2\mathbf{S} + 6\mathbf{A}$.

[5] We represent the costs of multiplications, squarings, and additions (or subtractions) in $\mathbb{F}_{p^2}$ as $\mathbf{M}$, $\mathbf{S}$ and $\mathbf{A}$, respectively.

Problem 1. Let $\chi(P) = (x_3, y_3) \in \mathbb{A}^2$ be such that P has order 3 in $E_{(a,c)}$. Let $E'_{(a',c')} = E_{(a,c)}/\langle P \rangle$, $\phi : E_{(a,c)} \to E'_{(a',c')}$, and $Q \in E_a \setminus \ker(\phi)$. The isogeny evaluation $\chi(\phi(Q)) = (x', y') \in \mathbb{A}^2$ from the additional input $\chi(Q) = (x, y) \in \mathbb{A}^2$ is done via the formula:

$$(x', y') = \left(\frac{x\left(x - \frac{1}{x_3}\right)^2}{(x - x_3)^2} x_3^2 : \frac{y\left(x - \frac{1}{x_3}\right)\left(\left(x - \frac{1}{x_3}\right)(x - x_3) + 2x\left(\frac{1}{x_3} - x_3\right)\right)}{(x - x_3)^3} x_3^2 \right).$$

Derive the respective projective degree-3 isogeny evaluation formula for computing $\chi(\phi(Q)) = (X' : Z') \in \mathbb{P}^1$ with the input $\chi(Q) = (X : Z) \in \mathbb{P}^1$, where $x = X/Z$ and $y = Y/Z$. Minimize the number of multiplications and squarings that are required, and write the final operation count.

3 Modular arithmetic

In this section, we describe efficient algorithms for computation over a quadratic extension field $\mathbb{F}_{p^2}$. We restrict the description to the most common setup for SIDH and SIKE, that is, we assume that $p \equiv 3 \bmod 4$ and fix $\mathbb{F}_{p^2} = \mathbb{F}_p(i)$ for $i^2 + 1 = 0$, where $p = 2^{e_2} \cdot 3^{e_3} - 1$.

3.1 Karatsuba multiplication meets lazy reduction

The $\mathbb{F}_{p^2}$ construction above allows us to leverage the extensive research done in the efficient implementation of such quadratic extension fields. In the context of pairings, high-speed implementations have exploited a combination of Karatsuba multiplication, lazy reduction and carry-handling elimination; e.g., see [2] for the implementation details of the multiplication over $\mathbb{F}_{p^2}$ used by the BN254 pairing. In particular, lazy reduction and carry-handling elimination are facilitated by using a prime with a bitlength that is slightly smaller than a multiple of a word size. If p is selected such that $l = \lceil \log_2 p \rceil < N$, where $N = n \cdot w$, $n = \lceil l/w \rceil$, and w is the computer wordsize, then several consecutive *integer* additions without modular correction and without carry-out in the most significant word (MSW) can be performed before a multiplication with the form $c = a \cdot b$, as long as the result c is guaranteed to be less than $2^N \cdot p$ (assuming the use of Montgomery reduction). Moreover, if the results of some integer multiplications (i.e., before reduction) are sufficiently smaller than $2^N \cdot p$, then lazy reduction could be applied and several consecutive double-precision additions without carry-outs in the MSW (and, in some cases, subtractions without borrow-outs in the MSW) can be performed before the Montgomery reduction. The algorithm for multiplication in $\mathbb{F}_{p^2}$ combining all these optimizations is shown in Algorithm 2. Integer operations without modular correction or reduction are represented as $\times, +$ or $-$. The only operation that requires a modular correction is the subtraction on line 4 that is represented as $\ominus$. Double-precision operands are represented in uppercase, while single-precision operands are in lowercase.

The SIKE primes [3] submitted to the NIST PQC process [16] were selected as to facilitate the techniques above. For example, the SIKE prime chosen for the NIST security level 1 is $p_{434} = 2^{216} \cdot 3^{137} - 1$. On a 32- and 64-bit platform, this prime has an extra room of $448 - \log_2 p_{434} = 14$ bits.

Algorithm 2 Multiplication in $\mathbb{F}_{p^2}$ using Karatsuba and lazy reduction

Input: $a = (a_0 + a_1 i)$ and $b = (b_0 + b_1 i) \in \mathbb{F}_{p^2}$
Output: $c = a \cdot b = (c_0 + c_1 i) \in \mathbb{F}_{p^2}$

```
1: $T_0 \leftarrow a_0 \times b_0,\ T_1 \leftarrow a_1 \times b_1,\ t_0 \leftarrow a_0 + a_1,\ t_1 \leftarrow b_0 + b_1$
2: $T_2 \leftarrow t_0 \times t_1,\ T_3 \leftarrow T_0 + T_1$
3: $T_2 \leftarrow T_2 - T_3$
4: $T_0 \leftarrow T_0 \ominus T_1$
5: $c_0 \leftarrow T_2 \bmod p$
6: $c_1 \leftarrow T_0 \bmod p$
7: **return** $C = (c_0 + c_1 i)$
```

3.2 Specialized modular reduction

Supersingular isogeny-based schemes have modular arithmetic at their core and, hence, their cost is dominated by the modular multiplication computations. A well-known method to implement this operation is due to Montgomery [15]. Let $R = 2^N$ and $p' = -p^{-1} \bmod R$, where again $N = n \cdot w$, $n = \lceil l/w \rceil$, $l = \lceil \log_2 p \rceil$ and w is the computer wordsize. Then, the Montgomery residue $c = aR^{-1} \bmod p$ for an input $a < pR$, can be computed as

$$c = (a + (ap' \bmod 2^N) \cdot p)/2^N, \tag{4}$$

which costs approximately $(n^2 + n)$ w-bit multiplications for a $2n$-limb value a.

Interestingly, the special shape of SIDH and SIKE primes is amenable for optimized versions of the so-called Montgomery reduction, as first noted by Costello et al. [5]. For a prime with the form $p = 2^{e_2} \cdot 3^{e_3} - 1$, the computation above simplifies to

$$\begin{aligned} c &= \left(a + (ap' \bmod 2^N) \cdot 2^{e_2} \cdot 3^{e_3} - (ap' \bmod 2^N)\right)/2^N \\ &= \lfloor \left(a + (ap' \bmod 2^N \cdot 2^{e_2} \cdot 3^{e_3}\right)/2^N \rfloor. \end{aligned} \tag{5}$$

In addition, since $p' = -p^{-1} \bmod 2^N$ also has a special form the cost of computing $ap' \bmod 2^N$ can be simplified further (e.g., $p' - 1$ for $p_{751} = 2^{372} \cdot 3^{239} - 1$ contains five 64-bit limbs or eleven 32-bit limbs of value 0). In total, the cost of computing c reduces to $n(n - \lfloor e_2/w \rfloor)$ w-bit multiplications. For example, if $w = 64$ (i.e., $n = 12$ for p_{751}), the theoretical speedup of the simplified modular reduction over p_{751} is about 1.86x.

Eliminating the Montgomery conditional subtraction. The result of a Montgomery reduction is upper bounded by $2p$ when its input is in the range $[0, pR)$. Hence, a conditional subtraction is needed to bring the result to $[0, p)$. However, this operation can be avoided if we perform arithmetic over a redundant representation (e.g., over $\mathbb{Z}_{2p}$). For example, if operands are kept in the range $[0, 2p)$ such that the result of a multiplication is guaranteed to be $c = a \cdot b < 4p^2 \leq pR$ (i.e., it should hold that $R \geq 4p$), then the result of the Montgomery reduction is still going to be bounded by $2p$ but we will no longer require the modular correction. A simple correction is going to be required at the very end of the computations to bring the final result to the canonical range $[0, p)$.

Note that this optimization complements quite nicely the previous techniques discussed in this section. This makes sense since avoiding the Montgomery conditional subtraction is just a variant of the approach of avoiding modular corrections in other parts of the arithmetic.

Algorithms 3 and 4 are specialized variants of multiplication and squaring (resp.) in $\mathbb{F}_{p^2}$ that take advantage of the use of redundant representations, as described above. Note that Algorithm 3 also exploits the form of the SIDH/SIKE primes to use a slightly faster subtraction with conditional correction at lines 4–6. Also note that Algorithm 4 includes a subtraction with unconditional addition at line 1 that guarantees that inputs fed to the multipliers have a positive value.

Algorithm 3 Specialized multiplication in $\mathbb{F}_{p^2}$ with redundant representation.

Input: $a = (a_0 + a_1 i)$ and $b = (b_0 + b_1 i) \in \mathbb{F}_{p^2}$. Input coefficients are allowed to be in the range $[0, 2p)$. $N = n \cdot w$, $n = \lceil l/w \rceil$, $l = \lceil \log_2 p \rceil$ and w is the computer wordsize.
Output: $c = a \cdot b = (c_0 + c_1 i) \in \mathbb{F}_{p^2}$. Output coefficients are allowed to be in the range $[0, 2p)$.

```
1: $T_0 \leftarrow a_0 \times b_0,\ T_1 \leftarrow a_1 \times b_1,\ t_0 \leftarrow a_0 + a_1,\ t_1 \leftarrow b_0 + b_1$
2: $T_2 \leftarrow t_0 \times t_1,\ T_3 \leftarrow T_0 + T_1$
3: $T_2 \leftarrow T_2 - T_3$
4: $T_0 \leftarrow T_0 - T_1$
5: if $T_0 < 0$ then
6:     $T_0 \leftarrow T_0 + p \cdot 2^N$
7: $c_0 \leftarrow T_2 \bmod p$
8: $c_1 \leftarrow T_0 \bmod p$
9: return $C = (c_0 + c_1 i)$
```

Problem 2. On the SIDH protocol, an isogeny curve operation always precedes calls to the isogeny evaluation function. Assuming that the inputs to formula (2) are in the range $[0, 2p)$, determine all the input/output bounds for the $\mathbb{F}_{p^2}$ operations in formula (3). At the $\mathbb{F}_{p^2}$ level assume the use of Algorithms 3 and 4 for multiplications and squarings (resp.), and the use of additions and subtractions *without* conditional corrections (unconditional corrections are allowed). What bound should be in place for p with regard to R for the Montgomery reduction

Algorithm 4 Specialized squaring in $\mathbb{F}_{p^2}$ with redundant representation.

Input: $a = (a_0 + a_1 i) \in \mathbb{F}_{p^2}$. Input coefficients are allowed to be in the range $[0, 2p)$.
Output: $c = a^2 = (c_0 + c_1 i) \in \mathbb{F}_{p^2}$. Output coefficients are allowed to be in the range $[0, 2p)$.

```
1: $t_0 \leftarrow a_0 + a_1,\ t_1 \leftarrow a_0 - a_1 + 2p$
2: $t_2 \leftarrow a_0 + a_0$
3: $c_0 \leftarrow t_0 \times t_1 \bmod p$
4: $c_1 \leftarrow t_2 \times a_1 \bmod p$
5: **return** $C = (c_0 + c_1 i)$
```

to work correctly? What single change is required in Algorithm 4 to keep all inputs to $\mathbb{F}_{p^2}$ operations in a positive range?

Radix-r Montgomery reduction. Straight implementations of Montgomery reduction as given in eq. (4) and its simplified variant in eq. (5) demand heavy storage resources because they operate over the full input using the full modulus in a single pass. A more practical approach proposed by Dussé and Kaliski Jr. [8] processes the computation one digit at a time reducing with r at each iteration, in what is called the radix-r Montgomery reduction. In this case, one picks a radix r to represent the input and the modulus, fixes $p' = -p^{-1} \bmod r$, and computes $n = \lceil l/r \rceil$ iterations doing

$$c = (a + (ap' \bmod r) \cdot p)/r. \tag{6}$$

In a way, the original Montgomery reduction is simply a special case of the radix-r variant, if we set $r = 2^N$ following the description at the beginning of Section 3.2. But setting the radix to a smaller value brings several benefits, including the reduction of the register use pressure and the reduction of the number of memory accesses. For special primes like SIDH primes, there is also the possibility of enabling further simplifications if r is such that $p' = -p^{-1} \bmod r \equiv 1$.

The Round 1 implementation of SIKE for x64 platforms[6] used a variant of the radix-r Montgomery reduction where r was set to 2^{64} for a computer wordsize $w = 64$. Internally, multiplications were done using Comba (i.e., carried out in product-scanning form). Later on, Faz-Hernández et al. [9] showed that in some cases it is more efficient to use a larger value for the radix. Ultimately, the optimal value for r is platform-dependent, and depends on several factors such as the number of general-purpose registers available, the relative cost of memory reads/writes, the relative cost of multiplication and addition instructions, etc. A flexible version of the approach optimized for SIDH/SIKE primes is shown in Algorithm 5.

[6] https://github.com/microsoft/PQCrypto-SIDH/releases/tag/v2.0

Algorithm 5 Radix-r Montgomery reduction for a prime with the form $p = 2^{e_2} \cdot 3^{e_3} - 1$.

Input: an integer a s.t. $0 \le a < pR$, where $R = 2^{nw}$, $p = 2^{e_2} \cdot 3^{e_3} - 1$, $n = \lceil l/w \rceil$, $l = \lceil \log_2 p \rceil$, and w is the computer wordsize; the value $z = \lfloor e_2/w \rfloor$ represents the number of 0-value limbs in $(p+1)$, and the value $\hat{p} = (p+1)/2^{zw}$; radix $r = 2^{bw}$ s.t. $b \mid n$ and $0 < b \le z$.
Output: the Montgomery residue $c = a \cdot R^{-1} \bmod p$ s.t. $c \in [0, 2p)$.

```
1: $t \leftarrow a$
2: for $i = 1$ to $n/b$ do
3:     $q \leftarrow t \bmod 2^{bw}$
4:     $t \leftarrow \lfloor t/2^{bw} \rfloor + 2^{(z-b)w} q \cdot \hat{p}$
5: return $c \leftarrow t$
```

Algorithm 5 applies a reduction by $r = 2^{bw}$ at every iteration. As explained before, the computation is simplified by noting that $p' = -p^{-1} \bmod 2^{bw} \equiv 1$ (this holds since $0 < b \le z$, where z is the number of 0 w-bit digits of the value $p+1$). Taking eq. (6) as starting point, the core computation in Algorithm 5 is derived as follows

$$\begin{aligned} c &= (a + (ap' \bmod r) \cdot p)/r = (a + (a \bmod 2^{bw}) \cdot p)/2^{bw} \\ &= (2^{bw} \lfloor a/2^{bw} \rfloor + a \bmod 2^{bw} \cdot (p+1))/2^{bw} \\ &= \lfloor a/2^{bw} \rfloor + 2^{(z-b)w} a \bmod 2^{bw} \cdot (p+1)/2^{zw} \\ &= \lfloor a/2^{bw} \rfloor + 2^{(z-b)w} q \cdot \hat{p} \end{aligned}$$

The most expensive operation in Algorithm 5 is the computation $q \times \hat{p}$, which consists of a $b \times (n-z)$-digit multi-precision multiplication. Hence, it is crucial to pick a suitable value for the radix that optimizes this operation for a given platform. For example, on an x64 processor with support for carry-preserving instructions such as `mulx`, `adcx` and `adox`, this multiplication can be implemented efficiently using the schoolbook method, as shown in [9].

Problem 3. Modify Algorithm 5 to make it work for **any** integer b in $0 < b \le z$.

For some primes, it happens that $e_2/w \approx \lfloor e_2/w \rfloor + 1$. Noticing this, Bos and Friedberger [4] suggested to shift the value $\hat{p}$ to increase z and trade multiplications for shifting operations in the computation $q \times \hat{p}$. For example, for $p_{503} = 2^{250} \cdot 3^{159} - 1$ we get $z = 3$ for $w = 64$, and the operation $q \times \hat{p} = q \times (2^{58} \cdot 3^{159})$ consists of an 8×5-digit multiplication in Algorithm 5. But if instead we right-shift $\hat{p}$ to obtain the modified value $\overline{p} = 3^{159}$, the operation above is reduced to an 8×4-digit multiplication. Afterwards, the result can be easily converted to the correct value with a left-shift of 58 bits.

4 Cryptanalysis and parameter selection for SIKE

Parameters for SIDH, and later for SIKE, were initially selected by modeling the computational supersingular isogeny (CSSI) problem as a black-box claw finding problem [10,5,3]. Solving this problem with a meet-in-the-middle (MitM) attack has asymptotic exponential complexities $\mathcal{O}(p^{1/4})$ and $\mathcal{O}(p^{1/6})$ on classical and quantum computers, respectively [10]. Accordingly, the initial SIKE submission to the NIST PQC effort in 2017 [3] included the parameter sets SIKEp503, SIKEp751 and SIKEp964,[7] to match or exceed the computational resources required for key searches on AES128 (security level 1), AES192 (security level 3) and AES256 (security level 5), respectively [17]. This changed later on when Adj et al. [1] observed that the memory required by the MitM attack was unrealistic (also of complexity $\mathcal{O}(p^{1/4})$), and suggested to use instead the van Oorschot-Wiener (vOW) parallel collision finding algorithm [18] to select parameters. This work was followed up by complementary studies that extended the analysis to quantum algorithms [11], and that confirmed the results for the classical case [6]. Using the vOW algorithm as the *overall* best attack against CSSI, the SIKE team updated their parameter selection for Round 2 of the NIST PQC process[8], proposing SIKEp434, SIKEp503, SIKEp610 and SIKEp751 for NIST levels 1, 2, 3 and 5, respectively [3].

4.1 vOW on SIKE

Let $f : S \to S$ be a (pseudo-)random function on a finite set S. The vOW algorithm finds collisions $f(r) = f(r')$ for distinct values $r, r' \in S$. Define *distinguished points* as elements in S that have a distinguishing property that is easy to test, and denote by θ the proportion of points of S that are distinguished. The vOW algorithm proceeds by executing collision searches in parallel, where each search starts at a freshly chosen point $x_0 \in S$ and produces a trail of points $r_i = f(r_{i-1})$, for $i = 1, 2, \ldots$, until a distinguished point r_d is reached. We denote by w to the number of triples with the form (r_0, r_d, d) that can be stored in some shared memory that is available for the cryptanalysis, where each triple represents a distinguished point and its corresponding trail. Every time in a search that a distinguished point is reached, two cases arise: (i) if the respective memory address is empty or holds a triple with a distinct distinguished point, the new triple (r_0, r_d, d) is added to memory and a new search is launched with a new starting point r_0, or (ii) if the distinguished point in the respective address is a match, a collision was detected. This process is repeated until the so-called *golden* collision is detected, which is the one that solves the underlying CSSI problem. Since some function versions can have a low chance of hitting the right collision, the success probability is greatly improved by changing f after a certain period of time.

[7] The name of the parameter set is assembled by concatenating "SIKEp" and the bitlength of the underlying prime p.

[8] Note that there were no parameter changes for Round 3 compared to Round 2.

Heuristically, van Oorschot and Wiener determined that the total runtime of the algorithm is minimized when fixing $w \geq 2^{10}$ and $\theta = 2.25\sqrt{w/|S|}$, and the total number of distinguished points generated by each function version before it is replaced is set to $10w$. Under these conditions, the total runtime to find a golden collision is given by

$$T = \left(\frac{2.5}{m}\sqrt{|S|^3/w}\right)t, \tag{7}$$

where m is the number of search engines that are run in parallel by an attacker, and t, in the case of SIDH/SIKE, is the time for one run of a degree-$\ell^{e/2}$ isogeny.

As before, we have that $p = 2^{e_2} \cdot 3^{e_3} - 1$. For typical parameters, running vOW on SIKE is usually more efficient when run on the 2-torsion. In this case, we set $\ell = 2$. For an even e_2 the size of the search space is given by $|S| = 2^{e_2/2-1}$, while for an odd e_2 it is $|S| = 2^{(e_2-1)/2}$. Costello et al. [6] showed that it is possible to fit a memory unit containing a triple (r_0, r_d, d) in $\lceil(2\log|S| + \log 20)/8\rceil$ bytes.

4.2 Cost models to estimate SIKE's security

Cost using the RAM model. The random access machine (RAM) model is typically used in algorithmic complexity analysis to estimate the total cost of running a certain algorithm. Simple versions of this approach use query complexity in conjunction with the approximated "time" that it takes to run an iteration of the algorithm, where "time" can be expressed using different units, such as clock cycles or instructions.

In Table 1, we show the classical security estimates for SIKE with respect to the vOW algorithm and using the RAM model. The query complexity for vOW is obtained by setting $t = m = 1$ in eq. (7) and assuming the availability of a shared memory with capacity for $w = 2^{80}$ memory units. The time to run an iteration of the algorithm is taken as the number of x64 instructions that are executed in the computation of a degree-$2^{e_2/2}$ isogeny [6].

Alternative cost models. One problem with the RAM model used to estimate the cost of cryptanalyzing SIKE is that it ignores the significant cost of the memory resources that are required to run vOW. In the specific case of Round 2 and Round 3 parameters, the 2^{80} memory units assigned to w in Table 1 are assumed to be given for free [1,6]. In contrast, Longa et al. [12] suggested to use a budget-based cost model that takes into account the cost of both computing and memory resources to determine the cryptanalysis cost. The analysis in [12] showed that, when considering the significant cost of memory, the Round 2 and Round 3 SIKE parameters are much more conservative than previously believed. The reader is referred to [12] for complete details.

A more standard approach is to use the non-local gate model and assume that the cost of memory is either $\mathcal{O}(w^{1/2})$ or $\mathcal{O}(w^{1/3})$. For the non-local gate model the cost is estimated as

Table 1: Classical cost to run vOW on the 2-torsion for the Round 3 SIKE parameters using the RAM model [6]. Memory size is set to $w = 2^{80}$, set size is given by $|S| = 2^{e_2/2-1}$. Numbers are shown as the floor of their base-2 logarithms. The number of isogeny computations, #isog, represents vOW's query complexity and is computed by setting $m = t = 1$ in eq. (7). The numbers $i_{\mathbf{sum}}$ of instructions for each isogeny are taken from [6, Table 5]. The total number of instructions, **vOW**, is the product of #isog and $i_{\mathbf{sum}}$ and is an estimate of the lower bound on the number of gates required to solve the CSSI problem with vOW.

	$\|S\|$	#isog	$i_{\mathbf{sum}}$	**vOW**	Classical gate requirement (NIST)
SIKEp434	107	121	22	**143**	143
SIKEp503	124	147	23	**170**	146
SIKEp610	151	187	23	**210**	207
SIKEp751	185	238	24	**262**	272

$$\textit{query complexity} \times \textit{gates} \times \textit{memory}. \tag{8}$$

Table 2 details the SIKE security estimates for level 1 parameters using this model. SIKEp377 is a new parameter set proposed in [12] that was showed to match more closely the NIST security level 1 requirement.

Table 2: Classical cost to run vOW on the 2-torsion for security level 1 parameters using the non-local gate model. The set size is given by $|S| = 2^{e_2/2-1}$ for even e_2 and $|S| = 2^{(e_2-1)/2}$ for odd e_2. vOW's query complexity is calculated by setting $m = t = 1$ in eq. (7). The gate complexity is fixed to $341,300$ and $372,200$ gates for SIKEp377 and SIKEp434 (resp.), following the hardware implementation results from [12]. The cost of memory is assumed to be either $\mathcal{O}(w^{1/2})$ or $\mathcal{O}(w^{1/3})$. For the latter, to estimate the number of memory units w we assume a budget of 2^{64} US dollars, that a byte of memory costs 0.22×10^{-11} US dollars [12, Table 10], and that each memory unit occupies $\lceil (2 \log |S| + \log 20)/8 \rceil$ bytes [6]. The total security estimates are obtained using eq. (8). Numbers are shown as the floor of their base-2 logarithms.

Memory cost	SIKEp434	SIKEp377	Classical gate requirement (AES128)
$\mathcal{O}(w^{1/2})$	180	**162**	143
$\mathcal{O}(w^{1/3})$	164	**145**	

Problem 4. Using the non-local gate model with a memory cost of $\mathcal{O}(w^{1/3})$, find the most efficient SIKE parameter set that fulfills a low security target of at least 104 bits (i.e., a classical gate requirement of at least 2^{104}). Assume that the gate complexity of the half degree isogeny implementation is 2^{18} gates. To calculate w assume a budget of 2^{64} US dollars, that a byte of memory costs 0.22×10^{-11} US dollars, and that each memory unit occupies $\lceil (2 \log |S| + \log 20)/8 \rceil$. Make sure p is "well-balanced" for security reasons, i.e., for $p = 2^{e_2} \cdot 3^{e_3} - 1$, we should have $e_2 \approx e_3$ (say, it should hold that $|e_2 - e_3| \leq 5$).

References

1. Gora Adj, Daniel Cervantes-Vázquez, Jesús-Javier Chi-Domínguez, Alfred Menezes, and Francisco Rodríguez-Henríquez. On the cost of computing isogenies between supersingular elliptic curves. In Carlos Cid and Michael J. Jacobson Jr., editors, *Selected Areas in Cryptography - SAC 2018*, volume 11349 of *LNCS*, pages 322–343. Springer, 2019.
2. D. F. Aranha, K. Karabina, P. Longa, C. H. Gebotys, and J. López. Faster explicit formulas for computing pairings over ordinary curves. In *Advances in Cryptology – Eurocrypt 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 48–68. Springer, 2011.
3. Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Aaron Hutchinson, Amir Jalali, Koray Karabina, David Jao, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Geovandro Pereira, Joost Renes, Vladimir Soukharev, and David Urbanik. Supersingular Isogeny Key Encapsulation (SIKE), 2017–2021. Latest specification available at https://sike.org. Round 1 submission available at https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/submissions/SIKE.zip. Round 2 submission available at https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-2/submissions/SIKE-Round2.zip.
4. Joppe W. Bos and Simon Friedberger. Fast arithmetic modulo $2^{\mathrm{x}}\,\mathrm{p}^{\mathrm{y}} \pm 1$. In Neil Burgess, Javier D. Bruguera, and Florent de Dinechin, editors, *24th IEEE Symposium on Computer Arithmetic, ARITH 2017*, pages 148–155. IEEE Computer Society, 2017.
5. Craig Costello, Patrick Longa, and Michael Naehrig. Efficient algorithms for supersingular isogeny Diffie-Hellman. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016*, volume 9814 of *LNCS*, pages 572–601. Springer, 2016.
6. Craig Costello, Patrick Longa, Michael Naehrig, Joost Renes, and Fernando Virdia. Improved classical cryptanalysis of SIKE in practice. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *Public-Key Cryptography - PKC 2020*, volume 12111 of *LNCS*, pages 505–534. Springer, 2020.
7. Luca De Feo, David Jao, and Jérôme Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *Journal of Mathematical Cryptology*, 8(3):209–247, 2014.

8. S. R. Dussé and B. S. Kaliski Jr. A cryptographic library for the Motorola DSP56000. In *Advances in Cryptology – Eurocrypt'90*, volume 473 of *Lecture Notes in Computer Science*, pages 230–244. Springer, 1991.
9. Armando Faz-Hernández, Julio López Hernandez, Eduardo Ochoa-Jiménez, and Francisco Rodríguez-Henríquez. A faster software implementation of the supersingular isogeny Diffie-Hellman key exchange protocol. *IEEE Trans. Computers*, 67(11):1622–1636, 2018.
10. David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In Bo-Yin Yang, editor, *Post-Quantum Cryptography - PQCrypto 2011*, volume 7071 of *LNCS*. Springer, 2011.
11. Samuel Jaques and John M. Schanck. Quantum cryptanalysis in the RAM model: Claw-finding attacks on SIKE. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019*, volume 11692 of *LNCS*, pages 32–61. Springer, 2019.
12. Patrick Longa, Wen Wang, and Jakub Szefer. The cost to break SIKE: A comparative hardware-based analysis with AES and SHA-3. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021*, volume 12827 of *Lecture Notes in Computer Science*, pages 402–431. Springer, 2021.
13. Microsoft. SIDH Library v3.4. Available at https://github.com/Microsoft/PQCrypto-SIDH, 2015–2021.
14. P. L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of computation*, 48(177):243–264, 1987.
15. Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):pp. 519–521, 1985.
16. National Institute of Standards and Technology (NIST). Post-quantum cryptography standardization, 2017–2020. https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization.
17. National Institute of Standards and Technology (NIST). Submission requirements and evaluation criteria for the post-quantum cryptography standardization process, December, 2016. https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf.
18. Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, 1999.