# SIBC: A Python-3 library for designing and implementing efficient isogeny-based protocols

Francisco Rodríguez-Henríquez *[1,2]

[1]Cryptography Research Centre, Technology Innovation Institute, Abu Dhabi, United Arab Emirates
[2]Computer Science Department, CINVESTAV-IPN, Mexico City, Mexico

September 17, 2021

## 1 Introduction

Adj, Chi-Domínguez and Rodríguez-Henríquez posted in `https://github.com/JJChiDguez/sibc`,[1], a Python-3 library named Supersingular Isogeny-Based Cryptographic constructions (SIBC). SIBC is a software tool that permits the efficient design an implementation of a variety of isogeny-based cryptographic protocols and their main building blocks, including the CSIDH, B-SIDH, SIKE and *B-SIKE* key exchange protocols. For example, SIBC supports a combination of Vélu's and $\sqrt{\text{é}}$lu's formulas, along with an adaptation of the optimal strategies commonly used in SIDH/SIKE, to produce efficient implementations of several instantiations of CSIDH. Here efficiency is measured in terms of the required number of field arithmetic operations. SIBC aims to provide fellow *isogenistas* with an agile design tool for constructing and testing new isogeny-based primitives, while keeping a constant-time execution of their procedures.

So far, SIBC has been used as a crucial software tool for fine-tuning and experimenting with the algorithms presented in several papers recently produced within our research group and collaborators [9, 12, 2, 10, 11].

The notes presented in this document are largely based on those papers.

### 1.1 Main building blocks

Generally speaking, performing isogeny map constructions and evaluations are the most expensive computational tasks of any isogeny-based protocol. Concretely, the main building blocks required for the implementation of isogeny-based cryptography are,

*francisco@cs.cinvestav.mx, francisco.rodriguez@tii.ae

[1]See `https://github.com/JJChiDguez/sibc/blob/master/CONTRIBUTORS` and the acknowledgment section of this document for a complete list of contributors.

- **Three point ladder** $P+[k]Q$: Given the points $\chi(P)$, $\chi(Q)$, and $\chi(Q-P)$ such that $P, Q-P \notin \{\mathcal{O}, (0,0)\}$, a right-to-left Montgomery ladder algorithm [21, 16] can compute $\chi(P+[k]Q)$ at a per-step cost of one point addition (xADD) and one point doubling (xDBL) operations, which are usually performed in the projective space $\mathbb{P}^1$.[2] The costs of one point addition and one point doubling are about the same for SIDH and CSIDH.

- **Optimal strategies**: Optimal strategies were introduced in [17] for minimizing the cost of the isogeny computations associated to SIDH. They permit to compute degree-$\ell^e$ isogenies at a cost of approximately $\frac{e}{2}\log_2 e$ scalar multiplications by $\ell$, $\frac{e}{2}\log_2 e$ degree-$\ell$ isogeny evaluations, and $e$ constructions of degree-$\ell$ isogenous curves. They apply equally well for CSIDH [20, 12], B-SIDH and B-SIKE [2].

- **Differential addition chains**: In the CSIDH protocol, any given scalar $k$ is the product of a subset of the collection of small primes $\ell_i$ dividing $\frac{p+1}{4}$. Hence, one can simply compute the scalar multiplication operation $[k]P$ as the composition of the shortest differential addition chains for each prime $\ell$ dividing $k$. Montgomery ladders using differential addition chains can perform the scalar multiplication operation $[k]P$ at an average length of about $1.5\lceil\log_2(k)\rceil$ steps [9]. Each Montgomery ladder step involves the computation of one differential point addition and differential point doubling at a cost of $4\mathbf{M}+2\mathbf{S}+6\mathbf{a}$ and $4\mathbf{M}+2\mathbf{S}+4\mathbf{a}$, respectively.

- **Vélu**: Since 1974, Vélu's formulas (cf. [22, §2.4] and [29, Theorem 12.16]) have been widely used to construct and evaluate degree-$\ell$ isogenies. Using several elliptic curve and isogeny arithmetic optimization tricks reported in the last few years [26, 14, 9]. The construction and evaluation of degree-$\ell$ isogenies via Vélu's formulas can be obtained at a computational cost of roughly $6\ell$ field multiplications

- **$\sqrt{\text{é}}$lu**: Using a baby-step giant-step approach, Bernstein, De Feo, Leroux and Smith presented in [6] a new approach for constructing and evaluating degree-$\ell$ isogenies at a combined cost of just $\tilde{O}(\sqrt{\ell})$ field operations. $\sqrt{\text{é}}$lu was extensively discussed in Week 10.

### 1.2 Isogeny-based key exchange protocols

In this isogeny-based cryptography school, we have revised a number of isogeny-based protocols and building blocks (often, the speakers have been the co-authors of the schemes presented in their lectures and notes), including:

[2] Points are mapped to $\mathbb{P}^1$ using:

$$\begin{array}{rcl} \chi\colon \ \mathbb{P}^2 & \mapsto & \mathbb{P}^1 \\ (X,Y,Z) & \to & (X,Z) \\ \mathcal{O} & \to & (1,0) \end{array}$$

- **SIDH**: Jao and De Feo presented in [21] (see also [17]) the Supersingular Isogeny-based Diffie-Hellman key exchange protocol (SIDH). SIDH operates with supersingular elliptic curves defined over the finite field $\mathbb{F}_{p^2}$, with $p$ a prime of the form $p = 2^{e_A}3^{e_B} - 1$. Different aspects of SIDH were discussed in this School during the Weeks 3-6 and also this Week 11.
- **SIKE**: In 2017, the Supersingular Isogeny Key Encapsulation (SIKE) protocol, an SIDH variant, was submitted to the NIST post-quantum cryptography standardization project [3]. On July 2020, NIST announced that SIKE passed to the round 3 of this contest as an alternate candidate. The efficient constant-time implementation of SIKE has been covered during this Week 11.
- **CSIDH**: In 2018, the commutative group action protocol CSIDH was introduced by Castryck, Lange, Martindale, Panny and Renes in [8]. CSIDH operates with supersingular elliptic curves defined over a prime field $\mathbb{F}_p$, and is a significantly faster version of the Couveignes-Rostovtsev-Stolbunov scheme variant that was presented in [18]. The efficient constant-time implementation of CSIDH has been covered during this Week 11.
- **B-SIDH**: In 2019, Costello proposed a variant of SIDH named B-SIDH [13]. In B-SIDH, Alice computes isogenies from a $(p+1)$-torsion supersingular curve subgroup, whereas Bob has to operate on the $(p-1)$-torsion subgroup of the quadratic twist of that curve. B-SIDH can achieve similar classical and quantum security levels as SIDH, but using [significantly] smaller public/private key sizes. On the other hand, B-SIDH requires the computation of isogenies of considerable large degree.

# 2 Traditional Vélu and its square root version

A Montgomery curve [25] is defined by the equation $E_{A,B} : By^2 = x^3 + Ax^2 + x$, such that $B \neq 0$ and $A^2 \neq 4$. For the sake of simplicity, we will write $E_A$ for $E_{A,1}$ and will always consider $B = 1$. Moreover, it is customary to represent the constant $A$ in the projective space $\mathbb{P}^1$ as $(A' : C')$, such that $A = A'/C'$ (see [15]).

Let $q = p^n$, where $p$ is a large prime number and $n$ a positive integer. Let $E$ be a supersingular Montgomery curve $E : y^2 = x^3 + Ax^2 + x$ defined over $\mathbb{F}_q$, and let $\ell$ be an odd prime number. Given an order-$\ell$ point $P \in E(\mathbb{F}_q)$, the construction of a degree-$\ell$ isogeny $\phi : E \mapsto E'$ of kernel $G = \langle P \rangle$ and its evaluation at a point $Q \in E(\mathbb{F}_q) \backslash G$ consists of the computation of the Montgomery coefficient $A' \in \mathbb{F}_q$ of the codomain curve $E' : y^2 = x^3 + A'x^2 + x$ and the image point $\phi(Q)$, respectively. In these notes, we will refer to these two tasks as isogeny construction and isogeny evaluation computations, respectively.

## 2.1 Vélu

Vélu's formulas (see [22, §2.4] and [29, Theorem 12.16]), have been generally used to construct and evaluate degree-$\ell$ isogenies by performing three main building blocks known

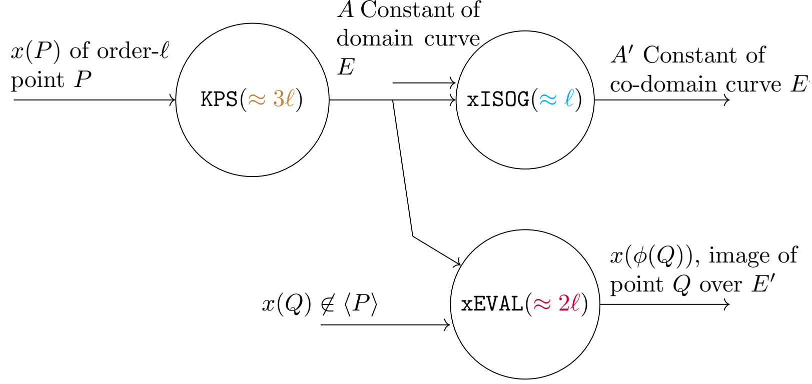


Figure 1: Traditional Vélu's formulas for coumputing the isogeny construction and isogeny evaluation operations. As shown in this figure, Vélu uses three main buliding blocks: `KPS`, `xEVAL` and `xISOG`

as, `KPS`, `xISOG` and `xEVAL`(see Figure 1).

The block `KPS` computes the first $k$ multiples of the point $P$, namely, the set $\{P, [2]P, \ldots, [k]P\}$. Using `KPS` as a sort of pre-computation ancillary module, `xISOG` finds the constants $(A' : C') \in \mathbb{F}_q$ that determine the codomain curve $E'$. Also, using `KPS` as a building block, `xEVAL` calculates the image point $\phi(Q) \in E'$. After applying a number of elliptic curve arithmetic tricks [26, 14, 9], the associated computational costs of these three building blocks can be summarized as,

- `KPS`: For each $1 \leq i \leq k$ we let $(X_i : Z_i) = x([i]P)$, where $\langle P \rangle = \ker(\phi)$.
  Cost: $\approx 3\ell$.
- `xEVAL`: One can compute $(X' : Z') = x(\phi(Q))$ from $(X_Q : Z_Q) = x(Q)$ as,

$$X' = X_P \Big( \prod_{i=1}^{k} \left[ (X_Q - Z_Q)(X_i + Z_i) + (Z_Q + Z_Q)(X_i - Z_i) \right] \Big)^2$$

$$Z' = Z_P \Big( \prod_{i=1}^{k} \left[ (X_Q - Z_Q)(X_i + Z_i) - (Z_Q + Z_Q)(X_i - Z_i) \right] \Big)^2$$

  Cost: $\approx 2\ell$.
- `xISOG`: Let us suppose that $F$ is a subgroup of the twisted Edwards curve $E_{a,d}$ with odd order $\ell = 2s + 1$, $s > 1$. Let the points in $F$ be given in twisted Edwards $YZ$-coordinates as the set,

$$\{(Y_1 : Z_1), \ldots, (Y_s : Z_s)\}.$$

Then, there exists a degree-$\ell$ isogeny $\psi$ with kernel $F$ that takes us from the curve $E_{a,d}$ to the curve $E_{a',d'}$. The constants $a', d'$ can be computed as,

$$By = \prod_{i=1}^{s} Y_i; \quad Bz = \prod_{i=1}^{s} Z_i;$$
$$a' = a^{\ell} B_z^8; \quad d' = B_y^8 d^{\ell}. \tag{1}$$

Cost: $\approx \ell$.

Which gives an overall cost of about $6\ell$ multiplications for the combined cost of the isogeny construction and evaluation tasks.

### 2.2 New Vélu's formulas

In this subsection we present the $\sqrt{\text{é}}$lu algorithms. For more details check [6, 2, 4] and the notes given in this school during Weeks 10 and 11.

Let $E_A/\mathbb{F}_q$ be an elliptic curve defined in Montgomery form by the equation $y^2 = x^3+Ax^2+x$, with $A^2 \neq 4$. Let $P$ be a point on $E_A$ of odd prime order $\ell$, and $\phi : E_A \to E_{A'}$ a separable isogeny of kernel $G = \langle P \rangle$ and codomain $E_{A'}/\mathbb{F}_q : y^2 = x^3 + A'x^2 + x$.

Our main task here is to compute $A'$ and the $x$-coordinate $\phi_x(\alpha)$ of $\phi(Q)$, for a rational point $Q = (\alpha, \beta) \in E_A(\mathbb{F}_q) \setminus G$. As mentioned in [6] (see also [14], [24] and [27]), the following formulas allow to accomplish this task,

$$A' = 2\,\frac{1+d}{1-d} \quad \text{and} \quad \phi_x(\alpha) = \alpha^{\ell}\,\frac{h_S(1/\alpha)^2}{h_S(\alpha)^2}, \text{ where}$$

$$S = \{1, 3, \dots, \ell-2\}, \quad d = \left(\frac{A-2}{A+2}\right)^{\ell} \left(\frac{h_S(1)}{h_S(-1)}\right)^{8}, \text{ and}$$
$$h_S(X) = \prod_{s \in S} (X - x([s]P)).$$

From the above, we see that the efficiency of computing $A'$ and $\phi_x(\alpha)$ directly depends on the cost of evaluating the polynomial $h_S(X) = \prod_{s\in S}(X - x([s]P))$. A naive approach would compute $h_S(X)$ by performing $\#S - 1$ polynomial products. Alternatively, exploiting a baby-step giant-step strategy $\sqrt{\text{é}}$lu obtains a square root complexity speedup over a traditional Vélu approach. In the following, we briefly sketch this strategy.

Given $E_A/\mathbb{F}_q$ an order-$\ell$ point $P \in E_A(\mathbb{F}_q)$, and some value $\alpha \in \mathbb{F}_q$ we want to efficiently evaluate the polynomial, $h_S(\alpha) = \prod_i^{\ell-1}(\alpha - x([i]P))$. From Lemma 4.3 of [6],

$$(X - x(P+Q))(X - x(P-Q)) = X^2 + \frac{F_1(x(P), x(Q))}{F_0(x(P), x(Q))}X + \frac{F_2(x(P), x(Q))}{F_0(x(P), x(Q))}$$

where,

$$\begin{aligned} F_0(Z,X) &= Z^2 - 2XZ + X^2; \\ F_1(Z,X) &= -2(XZ^2 + (X^2 + 2AX + 1)Z + X); \\ F_0(Z,X) &= X^2Z^2 - 2XZ + 1. \end{aligned} \tag{2}$$

This suggests a rearrangement à la Baby-step Giant-step as,

$$h(\alpha) = \prod_{i\in\mathcal{I}} \prod_{j\in\mathcal{J}} (\alpha - x([i + s\cdot j]P))(\alpha - x([i - s\cdot j]P))$$

Now $h(\alpha)$ can be efficiently computed by calculating the resultants of polynomials of the form,

$$\begin{aligned} h_I &\leftarrow \prod_{x_i\in\mathcal{I}} (Z - x_i)) \in \mathbb{F}_q[Z] \\ E_J(\alpha) &\leftarrow \prod_{x_j\in\mathcal{J}} \left(F_0(Z,x_j)\alpha^2 + F_1(Z,x_j)\alpha + F_2(Z,x_j)\right). \end{aligned}$$

The most demanding operations of $\sqrt{}$élu require computing four different resultants of the form $\texttt{Res}_Z(f(Z), g(Z))$ for polynomials $f, g \in \mathbb{F}_q[Z]$. We can compute those four resultants using a remainder tree approach supported by carefully tailored Karatsuba polynomial multiplications. In practice, the computational cost of performing degree-$\ell$ isogenies using $\sqrt{}$élu is close to $K(\sqrt{\ell})^{\log_2 3}$ field operations, with $K$ a constant [2].

#### 2.2.1 Concrete algorithms for KPS, xISOG, and xEVAL

As in section 2, we consider the three building blocks KPS, xISOG, xEVAL, where KPS computes the $x$-coordinates of all the points in the kernel $G$, xISOG finds the codomain coefficient $A'$, and xEVAL performs the computation of $\phi_x(\alpha)$.

In traditional Vélu, KPS calculates the product of the $x$-coordinates of ($\#S = (\ell - 1)/2$) points in the kernel $G$. As we have seen in Figure 1, this costs about $3\ell$ field multiplications. More efficiently, $\sqrt{}$élu only computes the $x$-coordinates of points of $G$ with indices in three subsets of $S$, each of size $O(\sqrt{\ell})$. Denote by $\mathcal{I}$, $\mathcal{J}$ and $\mathcal{K}$ those subsets of $S$. Then, $\mathcal{I}$ and $\mathcal{J}$ are chosen such that the maps $\mathcal{I}\times\mathcal{J} \to S$ defined by $(i,j) \mapsto i+j$ and $(i,j) \mapsto i-j$ are injective and their images $\mathcal{I}+\mathcal{J}$, $\mathcal{I}-\mathcal{J}$ are disjoint. The remaining indices of $S$ are gathered in $\mathcal{K} = S\backslash(\mathcal{I}\pm\mathcal{J})$.

Algorithm 1 computes KPS at a computational cost of about $\approx 3b$ point additions $= 18b\mathbf{M}$, and the storage of $\leq 8b$ field elements, where $b = \lfloor\sqrt{\ell-1}/2\rfloor$.

Let us recall that for the efficient computation of xISOG and xEVAL, $\sqrt{}$élu uses the biquadratic polynomials of Equation 2, which implies the computation of resultants of the form $\texttt{Res}_Z(f(Z), g(Z))$, for two polynomials $f, g \in \mathbb{F}_q[Z]$. For polynomials $f = a\prod_{0\leq i<n}(Z - x_i)$ and $g$ in $\mathbb{F}_q[Z]$, their resultant $\texttt{Res}(f,g) = a^n \prod_{0\leq i<n} g(x_i)$ can be computed efficiently when the factorization of $f$ is known, which is exactly our case.

**Algorithm 1** Kernel points computation (KPS)

**Require:** An elliptic curve $E_A/\mathbb{F}_q$; $P \in E_A(\mathbb{F}_q)$ of order an odd prime $\ell$.

**Ensure:** $\mathcal{I} = \{x([i]P) \mid i \in I\}$, $\mathcal{J} = \{x([j]P) \mid j \in J\}$, and $\mathcal{K} = \{x([k]P) \mid k \in K\}$ such that $(I, J)$ is an index system for $S$, and $K = S\backslash(I \pm J)$

1: $b \leftarrow \lfloor\sqrt{\ell - 1}/2\rfloor$; $b' \leftarrow \lfloor(\ell - 1)/4b\rfloor$
2: $I \leftarrow \{2b(2i + 1) \mid 0 \leq i < b'\}$
3: $J \leftarrow \{2j + 1 \mid 0 \leq j < b\}$
4: $K \leftarrow S\backslash(I \pm J)$
5: $\mathcal{I} \leftarrow \{x([i]P) \mid i \in I\}$
6: $\mathcal{J} \leftarrow \{x([j]P) \mid j \in J\}$
7: $\mathcal{K} \leftarrow \{x([k]P) \mid k \in K\}$
8: **return** $\mathcal{I}, \mathcal{J}, \mathcal{K}$

By means of a remainder tree approach, one evaluates one by one the factors $g(x_i)$ by computing $g \bmod (Z - x_i)$, $0 \leq i < n$. Then the product of all those factors gives us the resultant.

Thanks to the approach outlined above, the resultant $\texttt{Res}_Z(f(Z), g(Z))$ of two polynomials $f, g \in \mathbb{F}_q[Z]$ can be computed with an asymptotic runtime complexity of $\tilde{O}(n)$ by using a fast polynomial multiplication. Here fast means that this polynomial operation has a $O(n \log_2(n))$ field multiplication complexity (see [5, p. 7, §3]). The degree of the polynomials used for CSIDH and even B-SIDH, are sufficiently small so that Karatsuba polynomial multiplication (or related approaches such as Toom-Cook), emerges as the most efficient solution.Algorithm 2 and Algorithm 3 show the computation of the xISOG and xEVAL building blocks. All in all, the evaluation of KPS, xISOG, and xEVAL procedures have a combined computational cost of approximately [2, §4.3],

$$\begin{aligned}\text{Cost}(b) = {} & 4\left(3b^{\log_2(3)} + b\log_2(b) - \frac{5}{3}b + \frac{5}{6}\right) \\ & + \left(b^{\log_2(3)} - \frac{2}{3}b\right) + 2\left(3b^{\log_2(3)} - 2b\right) \\ & + 37b + 3\log_2(b) + 16 \\ = {} & 19b^{\log_2(3)} + 4b\log_2(b) + \frac{77}{3}b + 3\log_2(b) + \frac{58}{3}\,.\end{aligned} \tag{3}$$

One considerable advantage of using remainder trees here is that the subjacent product tree of the $(Z - x_i)$ factors, can be shared among all the resultants in Algorithm 2 and Algorithm 3, since these linear polynomials depend only on the kernel $\langle P\rangle$. In other words, the four resultants in Algorithm 2 and Algorithm 3 show no dependencies among them and therefore, they can be computed concurrently by a $\sqrt{\text{é}}$lu parallel implementation.

**Algorithm 2** Codomain curve construction (xISOG)

**Require:** An elliptic curve $E_A/\mathbb{F}_q : y^2 = x^3 + Ax^2 + x$; $P \in E_A(\mathbb{F}_q)$ of order an odd prime $\ell$; $\mathcal{I}, \mathcal{J}, \mathcal{K}$ from KPS.

**Ensure:** $A' \in \mathbb{F}_q$ such that $E_{A'}/\mathbb{F}_q : y^2 = x^3 + A'x^2 + x$ is the image curve of a separable isogeny with kernel $\langle P \rangle$.

1: $h_I \leftarrow \prod_{x_i \in \mathcal{I}}(Z - x_i)) \in \mathbb{F}_q[Z]$
2: $E_{0,J} \leftarrow \prod_{x_j \in \mathcal{J}} (F_0(Z, x_j) + F_1(Z, x_j) + F_2(Z, x_j)) \in \mathbb{F}_q[Z]$
3: $E_{1,J} \leftarrow \prod_{x_j \in \mathcal{J}} (F_0(Z, x_j) - F_1(Z, x_j) + F_2(Z, x_j)) \in \mathbb{F}_q[Z]$
4: $R_0 \leftarrow \texttt{Res}_Z(h_I, E_{0,J}) \in \mathbb{F}_q$
5: $R_1 \leftarrow \texttt{Res}_Z(h_I, E_{1,J}) \in \mathbb{F}_q$
6: $M_0 \leftarrow \prod_{x_k \in \mathcal{K}}(1 - x_k) \in \mathbb{F}_q$
7: $M_1 \leftarrow \prod_{x_k \in \mathcal{K}}(-1 - x_k) \in \mathbb{F}_q$
8: $d \leftarrow \left(\frac{A-2}{A+2}\right)^\ell \left(\frac{M_0 R_0}{M_1 R_1}\right)^8$
9: **return** $2\,\frac{1+d}{1-d}$

**Constant-time properties** Note that the procedures Algorithm 1, Algorithm 2 and Algorithm 3 compute $\sqrt{\text{é}}$lu in constant-time since,

- There are no branches with secret conditions.
- $\sqrt{\text{é}}$lu is a multiplicative-inverse-free procedure
- The three procedures KPS, xISOG and xEVAL are completely deterministic.
- The size of the sets $\mathcal{I}, \mathcal{J}$ and $\mathcal{K}$ as defined in KPS, are a function of the [public] parameter $\ell$.
- All the polynomial coefficients involved in the $\sqrt{\text{é}}$lu computation are different than zero. Hence, independently of the order-$\ell$ point $P$, the cost of the primitives KPS, xISOG and xEVAL is always the same.
- The remainder tree, which is at the heart of the two resultant computations, takes the same cost for either xISOG or xEVAL.
- Changing the kernel point $P$ does not affect the computational costs of KPS, xISOG and xEVAL.

**SIBC implementation** Using the option '-f', SIBC supports three formulas for computing isogenies: Vélu (tvelu), $\sqrt{\text{é}}$lu (svelu) and a hybrid version of Vélu and $\sqrt{\text{é}}$lu (hvelu).[3] See Figure 4 for command line examples.

[3]hvelu switches to the most efficient formula depending on the value of the degree-$\ell$ isogeny being computed.

**Algorithm 3** Isogeny evaluation (`xEVAL`)

**Require:** An elliptic curve $E_A/\mathbb{F}_q : y^2 = x^3 + Ax^2 + x$; $P \in E_A(\mathbb{F}_q)$ of order an odd prime $\ell$; the $x$-coordinate $\alpha \neq 0$ of a point $Q \in E_A(\mathbb{F}_q)\backslash\langle P\rangle$; $\mathcal{I}$, $\mathcal{J}$, $\mathcal{K}$ from `KPS`.

**Ensure:** The $x$-coordinate of $\phi(Q)$, where $\phi$ is a separable isogeny of kernel $\langle P\rangle$.

1: $h_I \leftarrow \prod_{x_i\in\mathcal{I}}(Z - x_i)) \in \mathbb{F}_q[Z]$

2: $E_{0,J} \leftarrow \prod_{x_j\in\mathcal{J}}\left(\frac{F_0(Z,x_j)}{\alpha^2} + \frac{F_1(Z,x_j)}{\alpha} + F_2(Z,x_j)\right) \in \mathbb{F}_q[Z]$

3: $E_{1,J} \leftarrow \prod_{x_j\in\mathcal{J}}\left(F_0(Z,x_j)\alpha^2 + F_1(Z,x_j)\alpha + F_2(Z,x_j)\right) \in \mathbb{F}_q[Z]$

4: $R_0 \leftarrow \mathtt{Res}_Z(h_I, E_{0,J}) \in \mathbb{F}_q$

5: $R_1 \leftarrow \mathtt{Res}_Z(h_I, E_{1,J}) \in \mathbb{F}_q$

6: $M_0 \leftarrow \prod_{x_k\in\mathcal{K}}(1/\alpha - x_k) \in \mathbb{F}_q$

7: $M_1 \leftarrow \prod_{x_k\in\mathcal{K}}(\alpha - x_k) \in \mathbb{F}_q$

8: **return** $(M_0R_0)^2/(M_1R_1)^2$

# 3 Loose notes on two isogeny-based key exchange protocols

Here, we give a simplified description of CSIDH and B-SIDH. For more technical details, the interested reader on CSIDH and B-SIDH is referred to [8, 9, 23, 28], and [13, 2], respectively.

## 3.1 Overviewing the C-SIDH

CSIDH is an isogeny-based protocol that can be used for key exchange and encapsulation [8], and other more advanced protocols and primitives. Figure 2 shows how CSIDH can be executed analogously to Diffie–Hellman, to produce a shared secret between Alice and Bob. Remarkably, the elliptic curves $E_{BA}$ and $E_{AB}$ computed by Alice and Bob at the end of the protocol are one and the same.

The flow of Figure 2 can be implemented in the SIBC library by executing the fragment shown in Figure 3.

CSIDH works over a finite field $\mathbb{F}_p$, where $p$ is a prime of the form

$$p = 4\prod_{i=1}^{n}\ell_i - 1$$

with $\ell_1, \ldots, \ell_n$ a set of small odd primes. For example, the original CSIDH article [8] defined a 511-bit $p$ with $\ell_1, \ldots, \ell_{n-1}$ the first 73 odd primes, and $\ell_n = 587$. This instantiation is commonly known as CSIDH-512.

The set of public keys in CSIDH is a subset of all supersingular elliptic curves in Montgomery form, $y^2 = x^3 + Ax^2 + x$, defined over $\mathbb{F}_p$. Since the CSIDH base curve $E$ is supersingular, it follows that $\#E(\mathbb{F}_p) = (p + 1) = 4\prod_{i=1}^{n}\ell_i$.

The input to the CSIDH class group action algorithm is an elliptic curve $E : y^2 = x^3 + Ax^2 + x$, represented by its $A$-coefficient, and an ideal class $\mathfrak{a} = \prod_{i=1}^{n}\mathfrak{l}_i^{e_i}$, represented by its list of secret exponents $(e_i, \ldots, e_n) \in [\![-m \mathrel{..} m]\!]^n$. The output is the $A$-coefficient

Public parameter:
$E/\mathbb{F}_p \colon By^2 = x^3 + Ax^2 + x,$

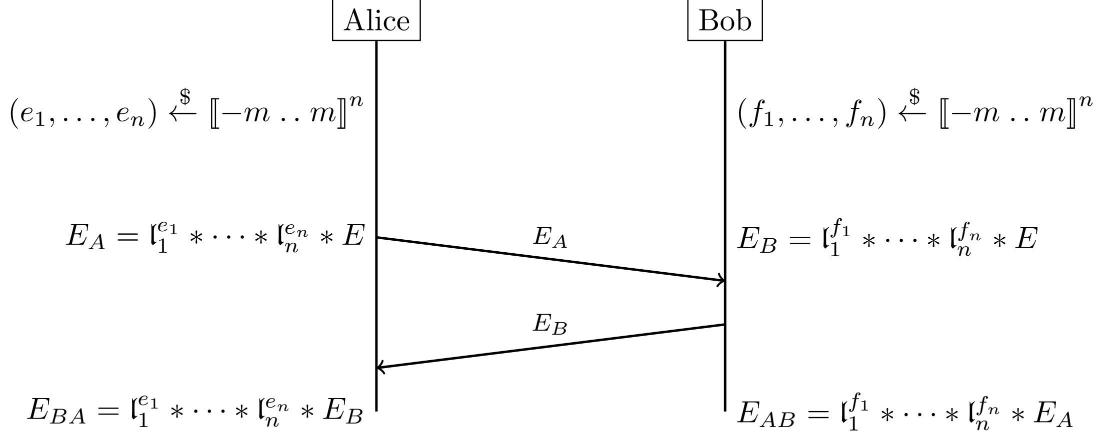


Figure 2: CSIDH key-exchange protocol

of the elliptic curve $E_A$ defined as,

$$E_A = \mathfrak{a} * E = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E. \tag{4}$$

Taking advantage of the commutative property of the group action, we can implement the protocol shown in Figure 2, which closely resembles the flow of the classical Diffie-Hellman protocol. Alice and Bob begin by selecting secret keys $\mathfrak{a}$ and $\mathfrak{b}$, and producing their corresponding public keys $E_A = \mathfrak{a} * E$ and $E_B = \mathfrak{b} * E$, respectively. After exchanging these public keys and taking advantage of the commutative property of the group action, Alice and Bob compute a shared secret as,

$$\mathfrak{a} * E_B = (\mathfrak{a} \cdot \mathfrak{b})E = (\mathfrak{b} \cdot \mathfrak{a})E = \mathfrak{b} * E_A.$$

#### 3.1.1 CSIDH variants

Our library supports three CSIDH variants, namely, the Meyer–Campos–Reith constant-time algorithm [23], the Onuki–Aikawa–Yamazaki–Takagi constant-time algorithm [28], and the dummy-free algorithm [9].

**One torsion point with dummy isogeny constructions (MCR-style)** Meyer, Campos and Reith proposed in [23] several ingenious optimizations that led to a fast constant-time CSIDH group action computation.

One of the optimizations introduced in [23], was to sample a point using the Elligator 2 map of [7]. A second optimization, dubbed SIMBA-$\sigma$-$\kappa$, consisted of splitting the processing of the prime factors $\ell_i$ as defined above, into $\sigma$ disjoint sets (batches) of size

```
from sibc.csidh import CSIDH, default_parameters
csidh = CSIDH(**default_parameters)

# alice generates a key
alice_secret_key = csidh.secret_key()
alice_public_key = csidh.public_key(alice_secret_key)

# bob generates a key
bob_secret_key = csidh.secret_key()
bob_public_key = csidh.public_key(bob_secret_key)

# if either alice or bob use their secret key with the other's respective
# public key, the resulting shared secrets are the same
shared_secret_alice = csidh.dh(alice_secret_key, bob_public_key)
shared_secret_bob = csidh.dh(bob_secret_key, alice_public_key)

# Alice and bob produce an identical shared secret
assert shared_secret_alice == shared_secret_bob
```

Figure 3: Executing the CSIDH key exchange of Figure 2 using SIBC

$\frac{n}{\sigma}$. Afterwards, a multiplicative strategy is applied to each batch. Each multiplicative strategy is evaluated $\kappa$ times. Finally, instead of using a fixed interval $[0, 10]$ for all the isogeny computations, the authors proposed to define a customized interval per each entry in the secret vector $e$. Thus, a vector $m$ is defined such that $0 \leq e_i \leq m_i$, for $i = 1, \ldots, n$. The missing prime factors are repaired using a multiplicative strategy, until all the $m_i$ degree-$\ell_i$ isogeny constructions have been performed.

**Two torsion point with dummy isogeny constructions (OAYT-style)** Onuki, Aikawa, Yamazaki and Takagi proposed a faster constant-time version of CSIDH in [28]. Their key idea is to use two points to evaluate the action of an ideal, one in $\ker(\pi - 1)$ (*i.e.*, in $E(\mathbb{F}_p)$) and one in $\ker(\pi + 1)$ (*i.e.*, in $E(\mathbb{F}_{p^2})$ with the $x$-coordinate in $\mathbb{F}_p$). This allows them to avoid timing attacks, while keeping the same primes and exponent range $[-5, 5]$ proposed in the original CSIDH algorithm [8]. Their algorithm also employs dummy isogenies to mitigate power analysis attacks, as in [23].

This way, the approach of [28] achieves a better performance than [23]. The speedup comes from the fact that the procedure proposed by [28] performs approximately five isogeny constructions (as opposed to the ten constructions in [23]) and ten isogeny evaluations per $\ell_i$.

**Two torsion point without dummy isogeny constructions (Dummy-free style)** In [9], the authors presented a constant-time CSIDH group action computation that does not use dummy computations. This gives some natural resistance to fault attacks, at the cost of approximately a twofold slowdown. For the approach in [9], the exponents $e_i$ are

```
# CSIDH
# A single random instances of a key exchange
sibc -p p512 -f hvelu -a csidh -s df -e 10 csidh-main
sibc -p p512 -f svelu -a csidh -s df -e 10 -m csidh-main
sibc -p p512 -f tvelu -a csidh -s df -e 10 -t csidh-main
sibc -p p512 -f hvelu -a csidh -s df -e 10 -m -t csidh-main
```

Figure 4: Executing a random dummy-free CSIDH key exchange of Figure 2 using a 512-bit prime $p$, a selection/combination of Vélu and Vélu sqrt for the isogeny computations and an integer range for the secret exponents between $[-10, 10]$.

uniformly sampled from sets

$$\mathcal{S}(m_i) = \{e \mid e = m_i \bmod 2 \text{ and } |e| \leq m_i\},$$

*i.e.*, centered intervals containing only even or only odd integers. The action of vectors drawn from $\mathcal{S}(m)^n$ can be computed by interpreting the coefficients $e_i$ as,

$$|e_i| = \underbrace{1 + 1 + \cdots + 1}_{e_i \text{ times}} + \underbrace{(1-1) - (1-1) + (1-1) - \cdots}_{m_i - e_i \text{ times}},$$

*i.e.*, the algorithm starts by acting by $\mathfrak{l}_i^{\mathtt{sign}(e_i)}$ for $e_i$ iterations, then alternates between $\mathfrak{l}_i$ and $\mathfrak{l}_i^{-1}$ for $m_i - e_i$ iterations.

**SIBC implementation** Using the option '-s', SIBC supports three CSIDH variants, namely, the Meyer–Campos–Reith constant-time algorithm of [23], the Onuki–Aikawa–Yamazaki–Takagi constant-time algorithm of [28], and the dummy-free algorithm of [9]. SIBC uses the following notation for the option '-s':

1. 'wd2': 'OAYT-style' (Two torsion point with dummy isogeny constructions)
2. 'wd1': 'MCR-style' (One torsion point with dummy isogeny constructions)
3. 'df': 'dummy-free-style' (Two torsion point without dummy isogeny constructions)

See Figure 4 for command line examples.

### 3.2 Playing the B-SIDH

B-SIDH was proposed by Costello in [13], Alice and Bob work in the $(p+1)$- and $(p-1)$-torsion of a set of supersingular curves defined over $\mathbb{F}_{p^2}$ and their quadratic twist set, respectively. B-SIDH is effectively twist-agnostic because optimized isogeny and Montgomery arithmetic only require the $x$-coordinate of the points along with the $A$

coefficient of the curve.[4] This feature implies that B-SIDH can be executed entirely *à la* SIDH as shown in Figure 5.[5]

More concretely, as before let $E : By^2 = x^3 + Ax^2 + x$ denote a supersingular Montgomery curve defined over $\mathbb{F}_{p^2}$, so that $\#E(\mathbb{F}_{p^2}) = (p+1)^2$, and let $E_t/\mathbb{F}_{p^2}$ denote the quadratic twist of $E/\mathbb{F}_{p^2}$. Then, $E_t/\mathbb{F}_{p^2}$ can be modeled as, $(\gamma B)y^2 = x^3+Ax^2+x$, where $\gamma \in \mathbb{F}_{p^2}$ is a non-square element and $\#E(\mathbb{F}_{p^2}) = (p-1)^2$. Notice that the isomorphism connecting these two curves is determined by the map $\iota\colon (x, y) \mapsto (x, jy)$ with $j^2 = \gamma$ (see [13, §3]).

Hence, for any $\mathbb{F}_{p^2}$-rational point $P = (x, y)$ on $E_t/\mathbb{F}_{p^2}$ it follows that $Q = \iota(P) = (x, jy)$ is an $\mathbb{F}_{p^4}$-rational point on $E$, such that $Q+\pi^2(Q) = \mathcal{O}$. Here $\pi\colon (x, y) \mapsto (x^p, y^p)$ is the Frobenius endomorphism. This implies that $Q$ is a zero-trace $\mathbb{F}_{p^4}$-rational point on $E/\mathbb{F}_{p^2}$.

B-SIDH can thus be seen as a reminiscent of the CSIDH protocol [8], where the quadratic twist is exploited to perform the computations using rational and zero-trace points with coordinates in $\mathbb{F}_{p^2}$. Although B-SIDH allows to work over smaller fields than either SIDH or CSIDH, it requires the computation of considerably larger degree-$\ell$ isogenies.

As illustrated in Figure 5, B-SIDH can be executed analogously to the main flow of the SIDH protocol. B-SIDH public parameters correspond to a supersingular Montgomery curve $E/\mathbb{F}_{p^2}\colon By^2 = x^3 + Ax^2 + x$ with $\#E(\mathbb{F}_{p^2}) = (p + 1)^2$, two rational points $P_a$ and $Q_a$ on $E/\mathbb{F}_{p^2}$, and two zero-trace $\mathbb{F}_{p^4}$-rational points $P_b$ and $Q_b$ on $E/\mathbb{F}_{p^2}$ such that

- $P_a$ and $Q_a$ are two independent order-$M$ points with $M \mid (p + 1)$, $\gcd(M, 2) = 2$, and $\left[\frac{M}{2}\right] Q_a = (0, 0)$;
- $P_b$ and $Q_b$ are two independent order-$N$ points with $N \mid (p-1)$ and $\gcd(N, 2) = 1$.

In practice, B-SIDH is implemented using projectivized $x$-coordinate points, and thus the point differences $PQ_a = P_a - Q_a$ and $PQ_b = P_b - Q_b$ must also be exchanged. Since the $x$-coordinates of $P_a, Q_a, PQ_a, P_b, Q_b$ and $PQ_b$, all belong to $\mathbb{F}_{p^2}$, a B-SIDH implementation must perform field arithmetic on that quadratic extension field. As in the case of SIDH, the protocol flow of B-SIDH must perform two main phases, namely, key generation and secret sharing. In the key generation phase, the evaluation of the projectivized $x$-coordinate points $x(P)$, $x(Q)$ and $x(P-Q)$ is required. Thus for B-SIDH, secret sharing is significantly cheaper than key generation.

The flow of Figure 5 can be implemented in the SIBC library by executing the fragment shown in Figure 6

Moreover, B-SIDH can naturally be extended to include a key encapsulation mechanism analogous to the extension of SIDH that was named SIKE. Thus, we adopt the

[4] For efficiency purposes, in practice both, the $x$-coordinate of the points and the constant $A$ of the curve, are projectivized to two coordinates.

[5] Although we omit here the specifics of the operations depicted in Figure 5, they are completely analogus to the ones corresponding to SIDH, a protocol that is carefully discussed in many papers such as [17, 15, 1].

Public parameter:
$E/\mathbb{F}_{p^2}\colon By^2 = x^3 + Ax^2 + x$,
$P_a, Q_a \in E[p+1]$ of order $M$, and $P_b, Q_b \in E[p-1]$ of order N

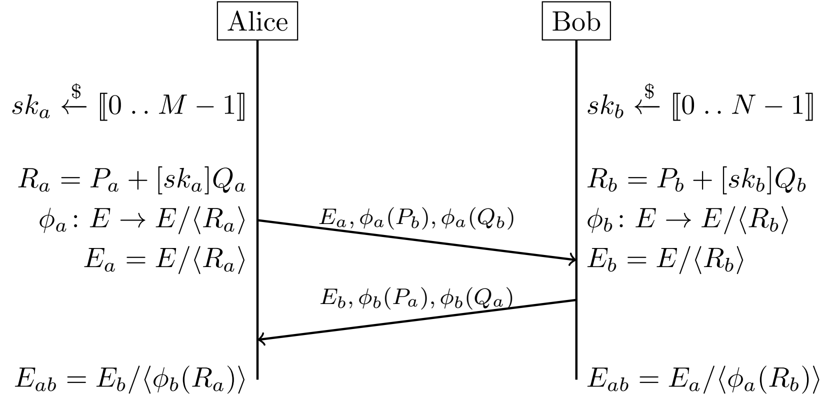


Figure 5: B-SIDH protocol for a prime $p$ such that $M|(p+1)$ and $N|(p-1)$.

```
from sibc.bsidh import BSIDH, default_parameters
bsidh = BSIDH(**default_parameters)
sk_a, pk_a = bsidh.keygen_a()
sk_b, pk_b = bsidh.keygen_b()
ss_a, ss_b = bsidh.derive_a(sk_a, pk_b), bsidh.derive_b(sk_b, pk_a)
ss_a == ss_b
```

Figure 6: Executing the B-SIDH key exchange of Figure 5 using SIBC

```
from sibc.sidh import SIKE, default_parameters
sike = SIKE(**default_parameters)
s, sk3, pk3 = sike.KeyGen()
c, K = sike.Encaps(pk3)
K_ = sike.Decaps((s, sk3, pk3), c)
K == K_

sike503 = SIKE('montgomery', 'p503', False, False)
s, sk3, pk3 = sike503.KeyGen()
c, K = sike503.Encaps(pk3)
K_ = sike503.Decaps((s, sk3, pk3), c)
K == K_
```

Figure 7: Executing a B-SIKE key exchange using SIBC and the prime $SIKE_p503$

name B-SIKE for the combination of B-SIDH plus a key encapsulation mechanism. The implemenantion in SIBC of B-SIKE, is shown in the fragment of Figure 7.

# 4 Frequently asked Questions

## 4.1 Library and programming related

0. How can I install SIBC in a Unix-like terminal?
   Use the following commands (For more details consult: `https://github.com/JJChiDguez/sibc/`):

```
# Installing required package
# Before running the following commands, ensure you have the lastest version of pip
pip3 install dh click numpy progress matplotlib networkx stdeb setuptools-scm setuptools

# only pip3 install
pip3 install pytest pytest-xdist

# For MAC/Window/LINUX use
pip install sibc

# Installing the library
sudo pip3 install -e .
```

1. In which language was SIBC written?
   SIBC is a Python-3 library. It is publicly available at: `https://github.com/JJChiDguez/sibc/`

2. Is there a user guide for SIBC?
   Once that the library is installed, automatically generated documentation is available via *pydoc*:

pydoc3 sibc.csidh
pydoc3 sibc.bsidh
pydoc3 sibc.sidh

3. Is there a quick help for the SIBC command syntax?
   Try: 'sibc –help'

4. Can I execute SIBC with my own primes?
   Yes. Check SIBC documentation for the exact syntaxis that you should use for defining arbitrary primes $p$ in the directory: 'data/sop/'.

5. Can I use SIBC to break any speed record for isogeny-based cryptographic scheme?
   While SIBC is a valuable tool for experimenting and testing new algorithmic tricks for isogeny-based cryptography, its execution timings are rather slow. After all, SIBC was written in Python3, which by no means produces fast executable code. However, since SIBC accurately reports the number of required field arithmetic operations, it is a useful testbench for assessing the performance of cryptographic schemes.

6. I have some ideas for improving some of the computations perform by SIBC, how can I test them and contribute with my improvements?
   To test your algorithmic tricks, just install SIBC with the option: 'sudo pip3 install -e .'. Additionally, you are more than welcome to directly contact any of the authors mentioned in the Acknowledgment section of this document for further feedback and collaboration.

### 4.2 Vélu and square root Vélu

0. Is it accurate that the asymptotic cost of $\sqrt{\text{é}}$lu is $\tilde{O}(\sqrt{\ell})$ as claimed by [6]?
   Yes. Using variants of the FFT multiplication along with Schönage's method for polynomial multiplication, the asymptotic cost of $\sqrt{\text{é}}$lu is indeed $\tilde{O}(\sqrt{\ell})$.

1. But then why is better in practice to use Karatsuba polynomial multplication for computing $\sqrt{\text{é}}$lu is $\tilde{O}(\sqrt{\ell})$?
   Due to the hidden constants in the Schönage-FFT polynomial multiplication, Karatsuba is a more economical approach for polynomials of degree less than $\approx 300$. Notice that it is always possible to combine these two approaches.

2. What is the expected impact of $\sqrt{\text{é}}$lu for SIDH or SIKE?
   None.

3. What is the expected impact of $\sqrt{\text{é}}$lu for CSIDH?
   In [4], the authors report significant accelerations for CSIDH-512 CSIDH-1024 using $\sqrt{\text{é}}$lu. In [2], the authors report significant accelerations for CSIDH-1792 using $\sqrt{\text{é}}$lu.

4. Where can I find a *concrete* cost analysis for $\sqrt{\text{é}}$lu?
   Check [2].

5. What is the expected impact of $\sqrt{\text{é}}$lu for B-SIDH?
   Huge. Arguably, B-SIDH is the big winner among all the isogeny-based protocols

6. What would be the most attractive projects on $\sqrt{\text{é}}$lu related topics from an algorithmic point of view?

   - To tune-up the sets $\mathcal{I}$, $\mathcal{J}$ and $\mathcal{K}$ of `KPS` to hopefully obtained a reduced cost on $\sqrt{\text{é}}$lu
   - To study more efficient ways to perform the [scaled] remainder tree associated to the computation of the resultants (See [5])
   - To study other polynomial multiplication methods (such as Toom-Cook)

### 4.3 Isogeny based protocols

0. I noticed that SIBC does not currently support any isogeny-based digital signature scheme: why is that so?
   Testing and implementing some of the proposed isogeny-based digital signature schemes in SIBC is part of our future work, and is definitely of our interest to develop these extensions for our library.

1. What is B-SIKE?
   B-SIDH can naturally be extended to include a key encapsulation mechanism analogous to the extension of SIDH that was named SIKE. In SIBC we use the name 'B-SIKE' for the combination of B-SIDH plus a key encapsulation mechanism based on the Fujisaki-Okamoto transform.

2. Can I implement radical isogenies using SIBC?
   Yes. SIBC was used in [11] for testing the performance of several radical isogenies variants.

3. Can I implement the Verifiable Delay function of [19] using SIBC?
   No. SIBC does not currently supports bilinear pairing computations, which are required in the construction of [19].

4. Is B-SIKE faster than SIKE?
   Only for some primes in Alice side. For details check out [2].

## 5 Exercises

We suggest the programming exercises listed below. Please notice that they are ordered by difficulty (we think): from easiest to hardest.

0. Provide large primes with a bitlength greater than two thousand bits for implementing large instances of CSIDH (See [10] for an explanation of why this project is worth the try)
1. Use SIBC to accurately estimate the computational cost of a parallel implementation of $\sqrt{\text{é}}$lu.
2. Use SIBC to accurately estimate the computational cost of a parallel implementation of B-SIDH and CSIDH using $\sqrt{\text{é}}$lu and hvelu (as defined in the library).
3. Modify SIBC to compute $\sqrt{\text{é}}$lu using Toom-Cook instead of Karatsuba.
4. Modify SIBC to implement CTIDH [4].
5. Modify SIBC to implement the Verifiable Delay function of [19].
6. Modify SIBC to implement any isogeny-based digital signature scheme of your choice

## 6 Acknowledgements

All the coauthors in [9, 12, 2, 10, 11] have had an active role on the design and crafting of SIBC. The main authors of SIBC are Gora Adj, Jesús-Javier Chi-Domínguez and, the author of this document. Jorge Chávez-Saab was actively involved in the design of several cryptographic modules of the library. Jacob Appelbaum and Leif Ryge are the main contributors for the implementation of a user-friendly interface for the library. Fabiola-Argentina Hernández-Torres designed the SIBC logo.

See `https://github.com/JJChiDguez/sibc/blob/master/CONTRIBUTORS` for a complete list of contributors.

## References

[1] G. Adj, D. Cervantes-Vázquez, J. Chi-Domínguez, A. Menezes, and F. Rodríguez-Henríquez. On the cost of computing isogenies between supersingular elliptic curves. In C. Cid and Michael J. Jacobson Jr., editors, *Selected Areas in Cryptography - SAC 2018 - 25th International Conference*, volume 11349 of *Lecture Notes in Computer Science*, pages 322–343. Springer, 2018.

[2] G. Adj, J.-J. Chi-Domínguez, and F. Rodríguez-Henríquez. Karatsuba-based square-root vélu's formulas applied to two isogeny-based protocols. Cryptology ePrint Archive, Report 2020/1109, 2020. `https://ia.cr/2020/1109`.

[3] R. Azarderakhsh, M. Campagna, C. Costello, L. D. Feo, B. Hess, A. Jalali, D. Jao, B. Koziel, B. LaMacchia, P. Longa, M. Naehrig, G. Pereira, J. Renes, V. Soukharev, and D. Urbanik. Supersingular isogeny key encapsulation. second round candidate of the nist's post-quantum cryptography standardization process, 2017. Available at: `https://sike.org/`.

[4] G. Banegas, D. J. Bernstein, F. Campos, T. Chou, T. Lange, M. Meyer, B. Smith, and J. Sotáková. CTIDH: faster constant-time CSIDH. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):351–387, 2021.

[5] D. J. Bernstein. Fast multiplication and its applications. *Algorithmic Number Theory*, 44:325–384, 2008.

[6] D. J. Bernstein, L. D. Feo, A. Leroux, and B. Smith. Faster computation of isogenies of large prime degree. *IACR Cryptol. ePrint Arch.*, 2020:341, 2020.

[7] D. J. Bernstein, M. Hamburg, A. Krasnova, and T. Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 967–980, 2013.

[8] W. Castryck, T. Lange, C. Martindale, L. Panny, and J. Renes. CSIDH: an efficient post-quantum commutative group action. In T. Peyrin and S. D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018, Part III*, volume 11274 of *Lecture Notes in Computer Science*, pages 395–427. Springer, 2018.

[9] D. Cervantes-Vázquez, M. Chenu, J. Chi-Domínguez, L. D. Feo, F. Rodríguez-Henríquez, and B. Smith. Stronger and faster side-channel protections for CSIDH. In P. Schwabe and N. Thériault, editors, *Progress in Cryptology - LATINCRYPT 2019*, volume 11774 of *Lecture Notes in Computer Science*, pages 173–193. Springer, 2019.

[10] J. Chávez-Saab, J. Chi-Domínguez, S. Jaques, and F. Rodríguez-Henríquez. The SQALE of CSIDH: square-root vélu quantum-resistant isogeny action with low exponents. *To appear in: J. Cryptogr. Eng.*

[11] J. Chi-Domínguez and K. Reijnders. Don't forget the constant-time in CSURF. *IACR Cryptol. ePrint Arch.*, page 259, 2021.

[12] J. Chi-Domínguez and F. Rodríguez-Henríquez. Optimal strategies for CSIDH. *Advances in Mathematics of Communications*, 2020. Preprint version: `https://eprint.iacr.org/2020/417`.

[13] C. Costello. B-SIDH: supersingular isogeny diffie-hellman using twisted torsion. In S. Moriai and H. Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - Proceedings, Part II*, volume 12492 of *Lecture Notes in Computer Science*, pages 440–463. Springer, 2020.

[14] C. Costello and H. Hisil. A simple and compact algorithm for SIDH with arbitrary degree isogenies. In T. Takagi and T. Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017, Part II*, volume 10625 of *Lecture Notes in Computer Science*, pages 303–329. Springer, 2017.

[15] C. Costello, P. Longa, and M. Naehrig. Efficient algorithms for supersingular isogeny Diffie-Hellman. In M. Robshaw and J. Katz, editors, *Advances in Cryptology – CRYPTO 2016*, pages 572–601, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[16] A. Faz-Hernández, J. C. López-Hernández, E. Ochoa-Jiménez, and F. Rodríguez-Henríquez. A faster software implementation of the supersingular isogeny diffie-hellman key exchange protocol. *IEEE Trans. Computers*, 67(11):1622–1636, 2018.

[17] L. D. Feo, D. Jao, and J. Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *J. Math. Cryptol.*, 8(3):209–247, 2014.

[18] L. D. Feo, J. Kieffer, and B. Smith. Towards practical key exchange from ordinary isogeny graphs. In T. Peyrin and S. D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018, Part III*, volume 11274 of *Lecture Notes in Computer Science*, pages 365–394. Springer, 2018.

[19] L. D. Feo, S. Masson, C. Petit, and A. Sanso. Verifiable delay functions from supersingular isogenies and pairings. In S. D. Galbraith and S. Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019, Proceedings, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 248–277. Springer, 2019.

[20] A. Hutchinson, J. T. LeGrow, B. Koziel, and R. Azarderakhsh. Further optimizations of CSIDH: A systematic approach to efficient strategies, permutations, and bound vectors. In M. Conti, J. Zhou, E. Casalicchio, and A. Spognardi, editors, *Applied Cryptography and Network Security - 18th International Conference, ACNS 2020, Part I*, volume 12146 of *Lecture Notes in Computer Science*, pages 481–501. Springer, 2020.

[21] D. Jao and L. D. Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In B. Yang, editor, *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*, volume 7071 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 2011.

[22] D. R. Kohel. *Endomorphism rings of elliptic curves over finite fields.* PhD thesis, University of California at Berkeley, The address of the publisher, 1996. Available at:`http://iml.univ-mrs.fr/~kohel/pub/thesis.pdf`.

[23] M. Meyer, F. Campos, and S. Reith. On lions and elligators: An efficient constant-time implementation of CSIDH. In J. Ding and R. Steinwandt, editors, *Post-Quantum Cryptography - 10th International Conference*, volume 11505 of *Lecture Notes in Computer Science*, pages 307–325. Springer, 2019.

[24] M. Meyer and S. Reith. A faster way to the csidh. In *INDOCRYPT 2018*, volume 11356 of *Lecture Notes in Computer Science*, pages 137–152. Springer, 2018.

[25] P. L. Montgomery. Speeding the pollard and elliptic curve methods of factorization. *Mathematics of computation*, 48(177):243–264, 1987.

[26] D. Moody and D. Shumow. Analogues of vélu's formulas for isogenies on alternate models of elliptic curves. *Math. Comput.*, 85(300):1929–1951, 2016.

[27] D. Moody and D. Shumow. Analogues of vélu's formulas for isogenies on alternate models of elliptic curves. *Mathematics of computation*, 85(300):1929–1951, 2016.

[28] H. Onuki, Y. Aikawa, T. Yamazaki, and T. Takagi. (short paper) A faster constant-time algorithm of CSIDH keeping two points. In N. Attrapadung and T. Yagi, editors, *14th International Workshop on Security, IWSEC 2019*, volume 11689 of *Lecture Notes in Computer Science*, pages 23–33. Springer, 2019.

[29] L. Washington. *Elliptic Curves: Number Theory and Cryptography, Second Edition.* Chapman & Hall/CRC, 2 edition, 2008.