On √élu's formulae and their applications to CSIDH and B-SIDH constant-time implementations

Gora Adj [*1], Jesús-Javier Chi-Domínguez [†2], and Francisco Rodríguez-Henríquez [‡3]

[1]Departament de Matemàtica, Universitat de Lleida, Spain
[2]Tampere University, Tampere, Finland
[3]Computer Science Department, CINVESTAV-IPN, Mexico City, Mexico

September 25, 2020

Abstract

At a combined computational expense of about 6ℓ field operations, Vélu's formulae are used to construct and evaluate degree-ℓ isogenies in the vast majority of isogeny-based primitive implementations. Recently, Bernstein, de Feo, Leroux and Smith introduced a new approach for solving this same problem at a reduced cost of just $\tilde{O}(\sqrt{\ell})$ field operations. In this work, we present a concrete computational analysis of these novel formulae, along with several algorithmic tricks that helped us to significantly reduce their practical cost. Furthermore, we report a Python-3 implementation of several instantiations of CSIDH and B-SIDH using a combination of the novel formulae and an adaptation of the optimal strategies commonly used in the SIDH/SIKE protocols. Compared to a traditional Vélu constant-time implementation of CSIDH, our experimental results report a saving of 5.357%, 13.68% and 25.938% base field operations for CSIDH-512, CSIDH-1024, and CSIDH-1792, respectively. Additionally, the first implementation of the B-SIDH scheme in the open literature is reported here.

1 Introduction

Isogeny-based cryptography was independently introduced by Couveignes [15], Rostovtsev and Stolbunov in [30, 31]. Since then, an ever increasing number of isogeny-based key-exchange protocols have been proposed. A selection of those protocols, especially relevant for this work, are briefly summarized below.

[*]gora.adj@udl.cat
[†]jesus.chidominguez@tuni.fi
[‡]francisco@cs.cinvestav.mx

Working with supersingular elliptic curves defined over $\mathbb{F}_{p^2}$, the Supersingular Isogeny-based Diffie-Hellman key exchange protocol (SIDH) was presented by Jao and de Feo in [20] (see also [16]). In 2017, the Supersingular Isogeny Key Encapsulation (SIKE) protocol, a SIDH descendent, was submitted to the NIST post-quantum cryptography standardization project [2]. NIST recently announced that SIKE passed to the round 3 of this contest as an "alternative candidate".

In 2018, the commutative group action protocol CSIDH was introduced by Castryck, Lange, Martindale, Panny and Renes in [8]. Operating with supersingular elliptic curves defined over $\mathbb{F}_p$, CSIDH is a significantly faster version of the Couveignes-Rostovtsev-Stolbunov scheme variant as it was presented in [17].

Later, in 2019, Costello proposed a variant of SIDH named B-SIDH [12]. 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. A remarkable feature of B-SIDH is that it can achieve similar classical and quantum security levels as SIDH, but using significantly smaller public/private key sizes. On the down side, at the time of writing, there has been no reported implementation of B-SIDH highlighting any potential benefit of its shorter key over its predecessors. The single most important challenge in the implementation of B-SIDH, is the high computational cost associated to the large degree isogenies involved in its execution.

Let ℓ be an odd prime number, $\mathbb{K}$ a finite field of large characteristic, and A a Montgomery coefficient of an elliptic curve $E : y^2 = x^3 + Ax^2 + x$. Given an order-ℓ point $P \in E(\mathbb{K})$, the construction of an isogeny $\phi : E \mapsto E'$ of kernel $\langle P \rangle$ and its evaluation at a point $Q \in E(\mathbb{K}) \backslash \langle P \rangle$ consist of the computation of the Montgomery coefficient $A' \in \mathbb{K}$ of the codomain curve $E' : y^2 = x^3 + A'x^2 + x$ and the image point $\phi(Q)$, respectively. Generally speaking, performing isogeny map constructions and evaluations are the most expensive computational tasks of any isogeny-based protocol. This is especially true for CSIDH and B-SIDH, where [extremely] large odd prime degree-ℓ isogenies come into play.

For decades now, Vélu's formulae (cf. [21, §2.4] and [32, Theorem 12.16]) has been widely used to construct and evaluate degree-ℓ isogenies. With the introduction of several elliptic curve arithmetic tricks [25, 13, 9], it turns out that Vélu's formulae require about 6ℓ field multiplications for the combined isogeny construction and evaluation procedures (cf. §2).

Recently, Bernstein, de Feo, Leroux and Smith presented in [4] a new approach for constructing and evaluating degree-ℓ isogenies at a combined cost of just $\tilde{O}(\sqrt{\ell})$ field operations. This improvement was obtained by observing that the polynomial product embedded in the isogeny computations can be speedup via a baby-step giant-step method [4, Algorithm 2]. Due to its square root complexity reduction (up to polylogarithm factors), in the remainder of this paper, we will refer to this improvement of Vélu's formulae computation as $\sqrt{}$élu's formulae or simply $\sqrt{}$élu.

As we will see in this paper, and as it was already hinted in [4], $\sqrt{}$élu has a high impact on the performance of CSIDH, and quite especially on B-SIDH. By way of illustration, consider the combined cost of constructing and evaluating degree-ℓ isogenies for $\ell = 587$,

which corresponds to an example highlighted in [4, Appendix A.3]. [1] For that degree ℓ, the authors report a cost of just $2296 \approx 3.898(\ell + 2)$ field multiplications and squaring operations. In this paper, we further improve that computation to just $2180 \approx 3.701(\ell+2)$ field multiplications and squaring operations, which is 5% cheaper than the cost reported by [4]. This has to be compared with the cost of a classical Vélu approach that would take some $3544 \approx 6.017(\ell + 2)$ multiplications.

In spite of the groundbreaking result announced in [4], along with the high performance achieved by its companion software library, the authors did not focus on providing a concrete computational cost analysis of their approach but rather, they centered on its asymptotical analysis. Moreover, an application of their fast √élu reported a rather modest 1% and 8% speedup over the traditional Vélu's formulae applied to the non constant-time implementation of the CSIDH instantiations, CSIDH-512 and CSIDH-1024. Furthermore, the authors of [4] left open the problem of assessing the practical impact of √élu on CSIDH and B-SIDH constant-time implementations.

Contributions. We present a detailed concrete analysis of √élu. From this analysis, we conclude that for virtually all practical scenarios, the best approach for performing the polynomial products associated to the isogeny arithmetic is achieved by nothing more than carefully tailored Karatsuba polynomial multiplications. The main practical consequence of this observation is that computing degree-ℓ isogenies with √élu has a concrete computational cost closer to $K(b^{\log_2(3)})$, where $b = \sqrt{\ell}$, and K is a constant. We also present several tricks that permit to save multiplications when performing the polynomial products involving the polynomials E_{J_0} and E_{J_1} (cf. §4). Additionally, we exploit the fact that for computing `xEVAL`, the polynomials E_{J_0} and E_{J_1} are the reciprocal of each other. These observations help us to construct and evaluate a degree-587 isogeny using only $2180\mathbf{M} \approx 3.701(\ell+2)$. This is about 5.3% cheaper than the same computation announced in [4]. This improvement also pushes the limit to $\ell = 89$, where computing degree-ℓ isogenies with √élu becomes more effective than traditional Vélu.

In a nutshell, our main practical contributions can be summarized as follows:

1. In practice, the computational cost of computing degree-ℓ isogenies using √élu, is closer to $K(\sqrt{\ell})^{\log_2 3}$ field operations, with K a constant.

2. We used the framework presented in [10] to apply optimal strategies à la SIDH to CSIDH while exploiting √élu. This allows us to present the first application of √élu to constant-time implementations of its CSIDH-512, CSIDH-1024, and CSIDH-1792 instantiations. A comparison with respect to CSIDH using Vélu's traditional formulae, reports savings of 5.357%, 13.68% and 25.938% field $\mathbb{F}_p$-operations for CSIDH-512, CSIDH-1024, and CSIDH-1792, respectively.

3. We report the first constant-time implementation of the protocol B-SIDH introduced in [12]. Using the framework of [10], optimal strategies à la SIDH are applied

[1] Note that $\ell = 587$ is the largest prime factor of $\frac{p+1}{4}$, where p is the prime used in the popular CSIDH-512 instantiation of the CSIDH isogeny-based protocol.

to B-SIDH while also taking advantage of √élu. As expected and hinted in [4], the experimental results for B-SIDH show a saving of up to 75% as compared with an implementation of this protocol using traditional Vélu's formulae.

Our software library is freely available at

`https://github.com/JJChiDguez/velusqrt` .

Outline. The remainder of this paper is organized as follows. In §2, traditional Vélu's formulae are described. A compact description of the B-SIDH and CSIDH protocols is also given. In §3, we briefly discuss the application of optimal strategies to CSIDH and B-SIDH. In §4, an explicit description of √élu's main building blocks `KPS`, `xEVAL`, and `xISOG` is presented. In addition, we discuss several √élu's algorithmic improvements in §4.2. The experimental results obtained from our software are reported and discussed in §5. We discuss CSIDH and B-SIDH in §5.1 and §5.2, respectively. Finally, our concluding remarks are drawn in §6.

Notation. **M**, **S**, and **A** denote the cost of computing a single multiplication, squaring, and addition (or subtraction) in the base field $\mathbb{F}_p$.

2 Background

The vast majority of the fastest isogeny-based constant-time protocol implementations, have adopted Montgomery and twisted Edwards curve models for their schemes. A Montgomery curve [24] 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 [14]).

Let $q = p^n$, where p is an odd prime number and n a positive integer. Let ℓ be an odd number $\ell = 2k + 1$, with $d \geq 1$. Also, let E and E' be two supersingular elliptic curves defined over $\mathbb{F}_q$, for which there exists a cyclic degree-ℓ isogeny $\phi : E \to E'$ defined over $\mathbb{F}_q$. This implies that there must exist an ℓ-order point $P \in E(\mathbb{F}_q)$ such that $\mathrm{Ker}(\phi) = \langle P \rangle$. Given the domain elliptic curve E and an ℓ-order point $P \in E(\mathbb{F}_q)$, we are interested in the problem of computing the co-domain elliptic curve E'. Given a point $Q \in E(\mathbb{F}_q)$ such that $Q \notin \mathrm{Ker}(\phi)$, we are also interested in the problem of finding $\phi(Q)$, *i.e.*, the image of the point Q over E'. In this paper these two tasks are named isogeny construction and isogeny evaluation computations, respectively.

Vélu's formula (see [21, §2.4] and [32, Theorem 12.16]), has been generally used to construct and evaluate degree-ℓ isogenies by performing three main building blocks, namely, `KPS`, `xISOG` and `xEVAL`. 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 co-domain curve E'. Also, using `KPS` as a building block, `xEVAL` calculates the image point $\phi(Q) \in E'$.

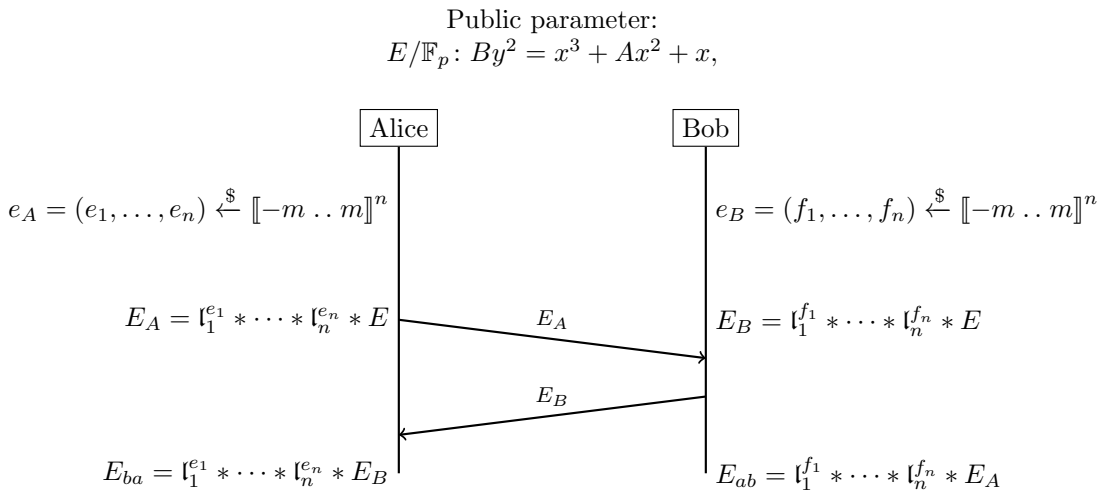


Figure 1: CSIDH key-exchange protocol

After applying a number of elliptic curve arithmetic tricks [25, 13, 9], the computational expenses of `KPS`, `xISOG` and `xEVAL` have been found to be about $3\ell, \ell$ and 2ℓ multiplications, respectively. This gives an overall cost of about 6ℓ multiplications for the combined cost of the isogeny construction and evaluation tasks. In §4, specific details of how the $\sqrt{}$élu approach of [4] drastically reduces the costs of traditional Vélu's formulae are discussed.

In the remainder of this section, we briefly discuss the two isogeny-based protocols implemented in this paper, namely, CSIDH and B-SIDH.

2.1 Overviewing the C-SIDH

Here, we give a simplified description of CSIDH. For more technical details, the interested reader is referred to [8, 9, 22, 27].

CSIDH is an isogeny-based protocol that can be used for key exchange and encapsulation [8], and other more advanced protocols and primitives. Figure 1 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.

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 $\#E(\mathbb{F}_p) = (p+1) = 4\prod_{i=1}^{n} \ell_i$.

Additionally, let $\pi\colon (x, y) \mapsto (x^p, y^p)$ be the Frobenius map and $N \in \mathbb{Z}$ be a positive integer. Then, $E[N] \coloneqq \{P \in E(\mathbb{F}_p)\colon [N]P = \mathcal{O}\}$ denotes the N-torsion subgroup of $E/\mathbb{F}_p$. Similarly, $E[\pi - 1] \coloneqq \{P \in E(\mathbb{F}_p)\colon (\pi - 1)P = \mathcal{O}\}$ and $E[\pi + 1] \coloneqq \{P \in E(\mathbb{F}_{p^2})\colon (\pi + 1)P = \mathcal{O}\}$ denote the subgroups of $\mathbb{F}_p$-rational and zero-trace points, respectively. In particular, any point $P \in E[\pi + 1]$ is of the form (x, iy) where $x, y \in \mathbb{F}_p$ and $i = \sqrt{-1}$ so that $i^p = -i$.

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 exponents $(e_i, \ldots, e_n) \in [\![-m \,..\, m]\!]^n$. The output, for Alice (See Figure 1), is the A-coefficient of the elliptic curve E_A defined as,

$$E_A = \mathfrak{a} * E = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E. \tag{1}$$

For the sake of simplicity, let us assume that the secret integer vector $e = (e_1, \ldots, e_n)$ is drawn from the interval $e_i \in [\![0 \,..\, m]\!]$. Let ϕ_{n-j} be a degree-ℓ_{n-j} isogeny defined as, $\phi_{n-j} : E_j \mapsto E_{(j+1) \bmod n}$, for $j = 0, \ldots, n-1$. Then, the CSIDH group action of Equation 1 can be computed as follows.

At the beginning of the group action evaluation, only the base elliptic curve $E_0 = E$ and the secret integer vector $e = (e_1, \ldots, e_n)$ are known. We then proceed by finding a full torsion point $T \in E_n[\pi - 1]$ (ideally) with order $\frac{p+1}{4} = \prod_i \ell_i$.[2]

Thereafter, for $j = 0, \ldots, n-1$, a subgroup kernel generator G_j is computed, and then the codomain of the corresponding degree-ℓ_{n-j} isogeny ϕ_{n-j} and the image point $\phi_{n-j}(T)$ are found. To obtain G_j, the point T must be *descended* by performing a scalar multiplication with the first $n-1-j$ prime factors of $p+1$. For example, for $j = 0$, the point $G_0 = \left[\prod_{i=1}^{n-1} \ell_i\right] T$ is computed. If G_0 is finite, then it has to have order ℓ_n and can be used to generate the kernel of the degree-ℓ_n isogeny ϕ_{ℓ_n}. Right after, the kernel subgroup $\langle G_j \rangle \leftarrow \texttt{KPS}(G_j)$, the image curve $E' = \texttt{xISOG}(E_j, \ell_{n-j}, \langle G_j \rangle)$ and the image point $T' = \phi_{n-j}(T) = \texttt{xEVAL}(T, \langle G_j \rangle)$ can all three of them be calculated. It becomes now possible to update the tuple (E_j, T, e_{n-j}) as,

$$(E_{(j+1) \bmod n}, T, e_{n-j-1}) \leftarrow \begin{cases} (E', T', e_{n-j-1}) \text{ if } e_{n-j} \neq 0; \\ (E_j, [\ell_{n-j}]T, e_{n-j-1}) \text{ otherwise,} \end{cases}$$

and $e_{n-j} \leftarrow \max\{0, e_{n-j} - 1\}$.

Once that all the n secret exponents e_j have been processed, the constants defining the elliptic curve E_0 are used to find a new full order point $T \in E_0$, restarting the procedure described above until *exactly* m evaluations are performed for all the secret exponents. This completes the CSIDH group action computation.

[2] In practice the computational cost required for finding a full-torsion point is too expensive. Therefore, this condition is relaxed to work with points whose order does not necessarily include all the prime factors of $p+1$. This leads to extra remedy steps not shown in Algorithm 4.

A constant-time procedure that performs the just described idealized strategy for computing the group action of Equation 1 is shown in Algorithm 4 ofAppendix A.

The computational cost of the group action is dominated by the calculation of n degree-ℓ_i isogeny evaluations and constructions plus a total of $\frac{n(n+1)}{2}$ scalar multiplications by the prime factors ℓ_i, for $i = 1, \ldots, n$. A similar multiplication-based approach for computing the group action algorithm was proposed in the original CSIDH protocol of [8]. It was first stated in [5, §8] (see also [19]) that this multiplication-based procedure could possibly be improved by adapting to CSIDH, the SIDH optimal strategy approach introduced by deFeo, Jao and Plût in [16]. We briefly discuss about the role of optimal strategies for large instances of CSIDH in §3, where the approach presented in [10] was adopted.

2.2 Playing the B-SIDH

In the B-SIDH protocol proposed by Costello in [12], 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 the set of their quadratic twist, 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.[3] This feature implies that B-SIDH can be executed entirely *à la* SIDH as shown in Figure 2.[4]

More concretely, 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 [12, §3]).

Hence, for any $\mathbb{F}_{p^2}$-rational point $P \coloneqq (x, y)$ on $E_t/\mathbb{F}_{p^2}$ it follows that $Q \coloneqq \iota(P) = (x, jy)$ 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-ℓ isogenies.

As illustrated in Figure 2, 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

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

[4] Although we omit here the specifics of the operations depicted in Figure 2, they are completely analogus to the ones corresponding to SIDH, a protocol that is carefully discussed in many papers such as [16, 14, 1].

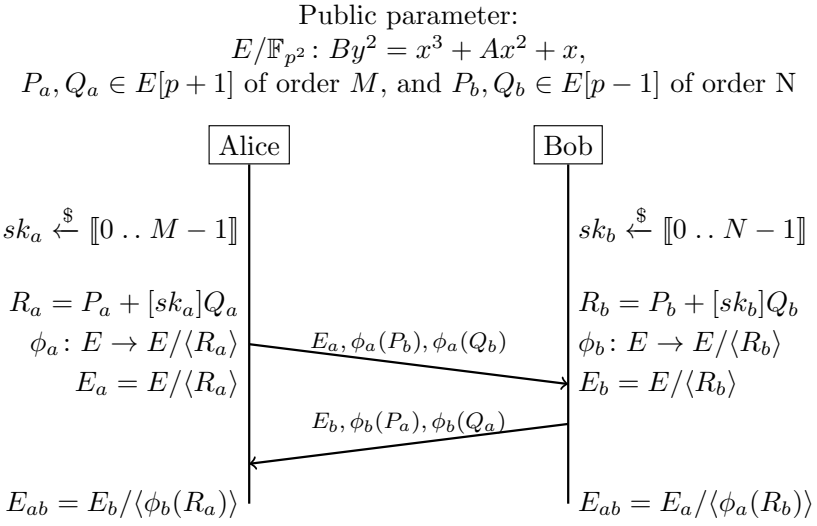


Figure 2: B-SIDH protocol for a prime p such that $M|(p+1)$ and $N|(p-1)$.

- 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 $PQ_a \coloneqq P_a - Q_a$ and $PQ_b \coloneqq P_b - Q_b$ must also be recorded. 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.

We briefly discuss the role of optimal strategies for large instances of B-SIDH in the next section.

3 Optimal strategies for the CSIDH and the B-SIDH

In [16], optimal strategies were introduced to efficiently compute degree-ℓ^e isogenies at a cost of approximately $\frac{e}{2}\log_2 e$ scalar multiplications by ℓ, $\frac{e}{2}\log_2 e$ degree-ℓ isogeny evaluations, and e constructions of degree-ℓ isogenous curves. Optimal strategies can be obtained using dynamic programming (see [2, 10] for concrete algorithms).

In the context of SIDH, optimal strategies tend to balance the number of isogeny evaluations and scalar multiplications to $O(n\log(n))$. However, CSIDH optimal strategies are expected to be largely multiplicative, *i.e.*, optimal strategies will tend to favor the

computation of more scalar multiplications. This is due to the fact that these operations are cheaper than large prime degree ℓ isogeny evaluations.

Let $L \coloneqq [\ell_1, \ell_2, \ldots, \ell_{74}]$ be the list of small odd prime numbers such that $p = 4 \cdot \prod_{i=1}^{n} \ell_i - 1$ is the prime number used in CSIDH. In this work we adopt the framework presented in [10], where the authors heuristically assumed that an arrangement of the set L from the smallest to the largest ℓ_i, is close to the global optimal. For this fixed ordering, it was presented in [10] a procedure that finds an optimal strategy with cubic complexity with respect to n.

Similarly to SIDH [16], optimal strategies can be used to improve the performance of B-SIDH, which requires the construction/evaluation of isogenies whose degrees are powers of large odd primes. In [19, 10], optimal strategies were applied to the context of CSIDH. In this work we adopted the framework proposed in [10], which permits an intuitive and easy integration of optimal strategies to B-SIDH. Let us assume that we need to construct a degree-L isogeny with $L \coloneqq {\ell_1}^{e_1} \cdot {\ell_2}^{e_2} \cdots {\ell_n}^{e_n}$, and let us write

$$L' = [\underbrace{\ell_1, \ldots, \ell_1}_{e_1}, \underbrace{\ell_2, \ldots, \ell_2}_{e_2}, \ldots, \underbrace{\ell_n, \ldots, \ell_n}_{e_n}]. \tag{2}$$

Then, an strategy for L' can be used to perform the key generation or secret sharing main phases of B-SIDH. In particular, any strategy for B-SIDH can also be encoded as in SIDH and CSIDH protocols, *i.e.*, by a list of $e-1$ positive integers where $e = \sum_{i=1}^{n} e_i$. Any such strategy can be evaluated from the procedure shown in Algorithm 5. As in SIDH [16] and CSIDH [10], optimal strategies are found by means of a dynamic-programming procedure. The evaluation of strategies for B-SIDH can be seen as an hybrid between SIDH and CSIDH. On the one hand, B-SIDH shares the same protocol flow with SIDH. On the other hand, B-SIDH must construct/evaluate multiple isogenies with degrees of powers of large odd primes as in CSIDH.

4 The new Vélu's formulae

This Section presents in more details the √élu algorithms when applied to isogeny-based cryptography. Several algorithmic tricks that slightly improve the performance of √élu as it was presented in [4] are given.

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 ℓ, 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 [4] (see also [13], [23] and [26]), the following formulae allow to accomplish this task,

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

$$\text{where } d = \left(\frac{A-2}{A+2}\right)^{\ell} \left(\frac{h_S(1)}{h_S(-1)}\right)^{8}, \quad S = \{1, 3, \ldots, \ell-2\}, \text{ and}$$

$$h_S(X) = \prod_{s \in S} (X - x([s]P)).$$

From this, one can see that the efficiency of computing A' and $\phi_x(\alpha)$ lies on that of computing $h_S(X)$. This is where $\sqrt{}$élu comes into play, with a baby-step giant-step strategy permitting a square root speedup over the traditional Vélu's formulae.

4.1 Construction and evaluation of odd degree isogenies

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

While the x-coordinates of $(\#S = (\ell - 1)/2)$ points in G are computed in `KPS` in the traditional Vélu algorithm, with the new formulae in [4] only the x-coordinates of points of G with indices in three subsets of S, each of size $O(\sqrt{\ell})$, are computed. 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. We call $(\mathcal{I}, \mathcal{J})$ an *index system for* S and write $\mathcal{I} \pm \mathcal{J}$ for $(\mathcal{I} + \mathcal{J}) \cap (\mathcal{I} - \mathcal{J})$. The remaining indices of S are gathered in $\mathcal{K} = S \backslash (\mathcal{I} \pm \mathcal{J})$. Algorithm 1 states the required `KPS` computations.

Algorithm 1 `KPS`

Require: An elliptic curve $E_A/\mathbb{F}_q$; $P \in E_A(\mathbb{F}_q)$ of order an odd prime ℓ.
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: $I \leftarrow \{2b(2i + 1) \mid 0 \le i < b'\}$
3: $J \leftarrow \{2j + 1 \mid 0 \le j < b\}$
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** $\mathcal{I}, \mathcal{J}, \mathcal{K}$

For the execution of `xISOG` and `xEVAL`, we need to define the following biquadratic polynomials:

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

The existence of these polynomials is a cornerstone of the √élu formulae. Indeed, they provide a way around to the non-homomorphicity of the x-coordinate map on elliptic curve points. We refer to [4] and [7, p. 132] for more details.

Let $\texttt{Res}_Z(f(Z), g(Z))$ denote the resultant of two polynomials $f, g \in \mathbb{F}_q[Z]$. We are now ready to outline `xISOG` and `xEVAL` in Algorithm 2 and Algorithm 3, respectively. Deriving the resultants in Algorithm 2 and Algorithm 3 may turn out to be a cumbersome task if it is not carried out in an elaborated way. 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 the case in the algorithms at hand. Employing a remainder tree approach (an equivalent alternative being continued fractions), one evaluates the factors $g(x_i)$ by computing $g \bmod (Z - x_i)$, $0 \leq i < n$, to take their product afterwards.

One considerable advantage of using remainder trees here is that the subjacent product tree of the $(Z - x_i)$ can be shared among all the resultants in Algorithm 2 and Algorithm 3, since these linear polynomials depend only on the kernel $\langle P\rangle$.

Algorithm 2 Computing `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 ℓ; $\mathcal{I}, \mathcal{J}, \mathcal{K}$ 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: $d \leftarrow \left(\frac{A-2}{A+2}\right)^{\ell} \left(\frac{M_0R_0}{M_1R_1}\right)^8$
9: **return** $2\,\frac{1+d}{1-d}$

Notice that the single most important high level operation is polynomial multiplication on the ring $\mathbb{F}_q[X]$. Thus, as deemed in [4], it is essential to utilize fast tailor-made polynomial multiplication algorithms, because in many places only a segment of the output product is needed. Certainly 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, where here fast means that it requires $O(n \log_2(n))$ field multiplications (see [3, p. 7, §3]). Nevertheless, the required degree polynomials for the case of CSIDH and even B-SIDH, are sufficiently small for karatsuba polynomial multiplication (or any of its variants like Toom-Cook), emerges as a more efficient solution. For example, according to the implementation of [4], $\ell = 587$ requires polynomials of degree $\#\mathcal{I} = 16$ and $2 \times \#\mathcal{J} = 18$ (in the B-SIDH case, $\#\mathcal{I}, \#\mathcal{J} \leq 150$). It can be easily

Algorithm 3 Computing `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 ℓ; 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 ϕ 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: $E_{0,J} \leftarrow \prod_{x_j \in \mathcal{J}} \left(F_0(Z, x_j)/\alpha^2 + F_1(Z, x_j)/\alpha + F_2(Z, x_j)\right) \in \mathbb{F}_q[Z]$
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** $(M_0R_0)^2/(M_1R_1)^2$

verified that Karatsuba polynomial multiplication becomes a more efficient choice.

4.2 Implementation speedups

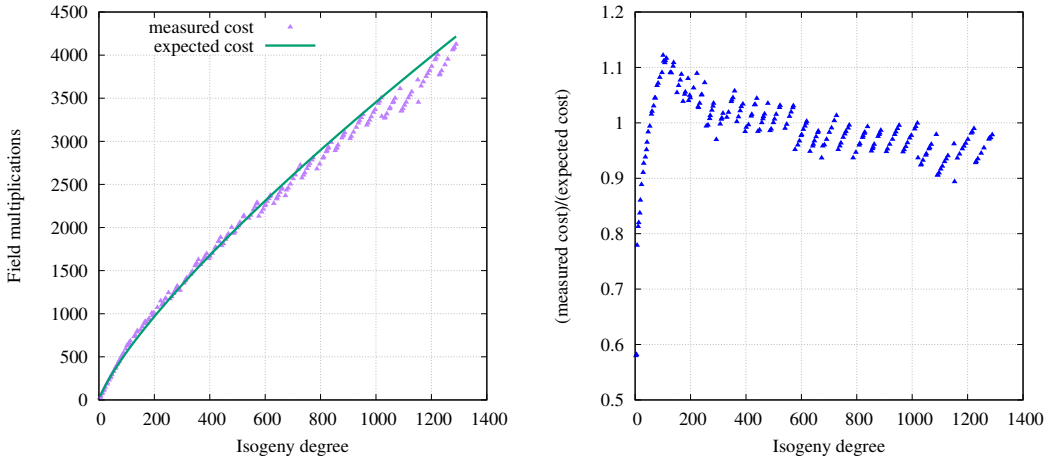
In this section we report a few algorithmic techniques that are exploited in our implementation to obtain some modest but noticeably savings over [4]. Our first refinement affects `xEVAL`, and arises from the special shape of the biquadratic polynomials F_0, F_1, F_2. In fact, with respect to either variable, one can see that F_1 is symmetric and F_0 is symmetric to F_2[5], that is, $F_1 = 1/Z^2\, F_1(1/Z, X)$ and $F_2 = 1/Z^2\, F_0(1/Z, X)$, considering the first variable for example. Now, using a projective representation of the x-coordinate $\alpha = x/z$ in `xEVAL`, we can write a quadratic polynomial factor in $E_{0,J}$ and a quadratic polynomial factor in $E_{1,J}$ respectively as

$$E_{0,j} = 1/x^2 \left(F_0(Z, x_j)z^2 + F_1(Z, x_j)xz + F_2(Z, x_j)x^2\right);$$
$$E_{1,j} = 1/z^2 \left(F_0(Z, x_j)x^2 + F_1(Z, x_j)xz + F_2(Z, x_j)z^2\right).$$

Thus, it becomes clear that the polynomials $x^{2\#J}E_{0,J}$ and $z^{2\#J}E_{1,J}$ are symmetric to one another, allowing to save the computation of one of the two products $E_{0,J}$, $E_{1,J}$. This gives us an expected saving of $\#\mathcal{J} \cdot \log_2(\#\mathcal{J})$ polynomial multiplications via product trees.

Our next improvement is focused on the computation of $E_{0,j}$. Let us write $x_j \coloneqq X_j/Z_j$.

[5] Consequently, all the quadratic factors of $E_{0,J}$ and $E_{1,J}$ in `xISOG` are symmetric. Bernstein et al. [4, Appendix A.5] were aware of this fact and took advantage of it to speed up the computation of $E_{0,J}$, $E_{1,J}$.



(a) Running time (b) Asymptotic constant

Figure 3: Measured and expected running time of `KPS` + `xISOG` + `xEVAL` for all the 207 small odd primes ℓ_i required in the group action evaluation of CSIDH-1792 (see [10]). All computational costs are given in $\mathbb{F}_p$-multiplications. The expected running time corresponds to $\mathrm{Cost}(b)$ with $b = \frac{\sqrt{(\ell-1)}}{2}$.

Then, $\left(F_0(Z,x_j)z^2 + F_1(Z,x_j)xz + F_2(Z,x_j)x^2\right)$ can be expressed as aZ^2+bZ+c, where

$$\begin{aligned} a &= C\cdot(x\cdot Z_j - z\cdot X_j)^2;\\ 2b &= \Big(\big[C\cdot(X^2+Z^2)\big]\cdot(-4\cdot X_j\cdot Z_j) - \big[2\cdot(X_j^2+Z_j^2)\big]\cdot\big(2\cdot[C\cdot(X\cdot Z)]\big)\Big)\\ &\quad + \Big(\big(2\cdot[A'\cdot(X\cdot Z)]\big)\cdot(-4\cdot X_j\cdot Z_j)\Big);\\ c &= (C\cdot(x\cdot X_j - z\cdot Z_j)^2. \end{aligned}$$

In fact, the three equations above, can be implemented (with the help of some extra pre-computations required in `xISOG`) at a cost of $7\mathbf{M} + 3\mathbf{S} + 12\mathbf{a}$ field operations. This cost should be compared with the implementation of [4], which requires $11\mathbf{M} + 2\mathbf{S} + 13\mathbf{a}$ field operations. Assuming $\mathbf{M} = \mathbf{S}$, this implies that our proposed formulae saves 3 field multiplications per polynomial $E_{0,j}$, $0 \le j < \#\mathcal{J}$.

Let us now illustrate the improvements just described applied to the example $\ell = 587$. Let us recall that in the implementation of [4], we have $\#\mathcal{I} = 16$ and $\#\mathcal{J} = 9$. Consequently, our first improvement saves $9\log_2(9) \approx 28$ polynomial multiplications via product trees. On the other hand, our second improvement saves $3 \times \#\mathcal{J} = 3 \times 9 = 27$ field multiplications.

4.3 Concrete computational analysis

In this subsection, the computational cost associated to the combined evaluation of the `KPS`, `xISOG`, and `xEVAL` procedures is derived.[6]

First note that `KPS` (see Algorithm 1), can be performed at a cost of about $3b$ differential point additions (assuming $\#\mathcal{I} \approx \#\mathcal{J} \approx \#\mathcal{K} \approx b$), which implies an expense of at most $(18b)\mathbf{M}$ field multiplications. Here $b = \lfloor \frac{\sqrt{\ell-1}}{2} \rfloor$ as given in Step 1 of Algorithm 1.

Observe also that the computation of the polynomial $h_I(Z)$ required at Step 1 of both, `xISOG` (Algorithm 2) and `xEVAL` (Algorithm 3) procedures, can be shared and thus must be computed only once. One interesting observation of [4], is that the computation of the polynomials $E_{0,J}$ and $E_{1,J}$ in `xISOG` (see Steps 2-3 of Algorithm 2), can be performed at a cost of only one product tree procedure. Furthermore, as it was already discussed in subsection 4.2, this same trick can also be applied to `xEVAL`, *i.e.*, Steps 2-3 of Algorithm 3 can be calculated by executing only one product tree. Hence, each polynomial $E_{i,J}$, $i = 0, 1$, required by `xISOG` and `xEVAL` can be obtained at a cost of $(3b)\mathbf{M}$ and $(10b)\mathbf{M}$ field operations, respectively.

Additionally, in Steps 4-5 of `xISOG` and `xEVAL`, the computation of two resultants are required, implying that four resultants must be computed in total. Each Resultant corresponds to the computation of $\mathtt{Res}_Z(f(Z), g(Z))$ such that $f, g \in \mathbb{F}_q[Z]$, $\deg f = b' \approx b$ and $\deg g = 2b$. A detailed description of the cost of computing such a resultant in terms of b by means of computing the leaves of remainder trees is given in Appendix B. In Appendix B, it is shown that the complexity in terms of field operations associated to the computation of a resultant as described in §4.2 is given as,

$$R(b) = 9b^{\log_2(3)} \left(1 - 2\left(\frac{2}{3}\right)^{\log_2(b)+1}\right) + 2b\log_2(b)\ . \tag{3}$$

The constants M_0 and M_1 in Steps 6-7 of `xISOG` and `xEVAL`, have a cost of $(2b)\mathbf{M}$ and $(4b)\mathbf{M}$ field operations, respectively. Lastly, the computations of the coefficient d of `xISOG` and the output of `xEVAL` require about $(3\log_2(b) + 16)$ multiplications. All in all and invoking Equation 3, the evaluation of `KPS`, `xISOG`, and `xEVAL` procedures have a combined cost of approximately,

$$\begin{aligned} \mathrm{Cost}(b) = 4 &\left(9b^{\log_2(3)} \left(1 - 2\left(\frac{2}{3}\right)^{\log_2(b)+1}\right) + 2b\log_2(b)\right) \\ &+ 3\left(\left(1 - \frac{1}{3^{\log_2(b)+1}}\right) b^{\log_2(3)}\right) \\ &+ 37b + 3\log_2(b) + 16. \end{aligned} \tag{4}$$

In order to verify the correctness of the cost predicted by Equation 4, the experiment described next was implemented.

[6] In the sequel, $\sqrt{}$élu computational costs are derived assuming a projective coordinate system and $\mathbf{M} = \mathbf{S}$.

We computed degree-ℓ isogenies for all the odd prime factors $\ell_1, \ell_2, \ldots, \ell_{207}$ of $p+1$, where p is the prime used in the CSIDH-1792 instantiation proposed in [10]. Figure 3 shows an excellent approximation between the theoretical cost of Equation 4 and the experimental results obtained from our Python3 software, where it was observed that (measured runtime) $\approx 0.97 \times$ (expected runtime).

Recall that the derivation of the expected cost of Equation 4 (See Appendix B), is driven by the assumption that $\mathbf{M} = \mathbf{S}$, which is the typical case for CSIDH. For the B-SIDH case on the other hand, since one is working on the quadratic extension field $\mathbb{F}_{p^2}$, it holds that $\mathbf{M} = 3\mathbf{M}_{\mathbb{F}_p}$ and $\mathbf{S} = 2\mathbf{M}_{\mathbb{F}_p}$, and thus $\mathbf{S} = \frac{2}{3}\mathbf{M}$. However, as an upper bound (for the B-SIDH case), we can assume $\mathbf{M} = 3\mathbf{M}_{\mathbb{F}_p}$ and $\mathbf{M} = \mathbf{S}$, which gives an expected running-time of $3 \times \text{Cost}(b)$ $\mathbb{F}_p$-multiplications.

A quick inspection of Algorithm 1-Algorithm 3, reveals that it is straightforward to concurrently compute many of the operations required by all three of those procedures. Specifically, the calculation of the four resultants in Steps 4-5 of Algorithm 2-Algorithm 3 show no dependencies among them and can therefore be computed in parallel by a multi-core processor. Since the four resultant calculations accounts for about 85% of the total computational cost of √élu, the expected savings are substantial.

5 Experiments and discussion

In this section we present a Python3-code constant-time implementation of the B-SIDH and CSIDH protocols, which make extensive usage of the √élu's formulae introduced in [4] boosted with the computational tricks presented in section 4. Furthermore, the optimal strategy framework presented in [10] is also exploited to maximize the performance of both protocols. Our software library is freely available at

`https://github.com/JJChiDguez/velusqrt` .

The main aim of our Python3-code software is to benchmark the total number of additions, multiplications, and squarings required by the instantiations of the two aforementioned protocols. To this end, we included counters inside the field arithmetic function cores for *fp_add()*, *fp_sub()*, *fp_mul()*, and *fp_sqr()*. Hence, all the performance figures presented in this section correspond with our count of field operations in the base field $\mathbb{F}_p$. In the case of the B-SIDH experiments, using standard arithmetic tricks the multiplication and squaring over $\mathbb{F}_{p^2}$ were performed at the cost of $3\mathbf{M} + 5\mathbf{a}$ and $2\mathbf{M} + 3\mathbf{a}$ base field operations, respectively.

All the experiments performed in this section are centered on comparing the following configurations, which are based on tradicional Vélu's formuale [13, 29] and √élu:

- Using the tradicional Vélu's formulae (labeled as *tvelu*);
- Using √élu (labeled as *svelu*);
- Using a hybrid between traditional Vélu and √élu (labeled as *hvelu*).

Notice that because of the nature of each protocol, the B-SIDH experiments are randomness-free, which implies that the same cost is reported for any given instance. In contrast, the CSIDH experiments have a variable cost determined by the randomness introduced by the order of the torsion points sampled from its Elligator-2 procedure (for a more detailed explanation see [9]).

5.1 Experiments on the CSIDH

Our Python3-code implementation of the CSIDH protocol includes a portable version for the following CSIDH instantiations,

1. Two torsion point with dummy isogeny constructions (OAYT-style [27])
2. One torsion point with dummy isogeny constructions (MCR-style [22])
3. Two torsion point without dummy isogeny constructions (Dummy-free style [9])

Our software supports performing experiments with any prime field of $p = 2^e \cdot (\prod_{i=1}^{n} \ell_i) - 1$ elements, for any $e \geq 1$. Our experiments were focused on the CSIDH-512 prime proposed in [8], the CSIDH-1024 prime proposed in [4], and the CSIDH-1792 prime proposed in [10]. The required number of field operations for those CSIDH variants are reported in Table 1, Table 2, and Table 3. In addition, each table presents a comparison between the results of this work and the ones presented in [10]. It is worth mentioning that optimal strategies and suitable bound vectors according to [10, section 3.4, 4.4 and 4.5] were used and computed for each configuration.

When comparing with respect to CSIDH constant-time implementations using traditional Vélu's formulae, our experimental results report a saving of 5.357%, 13.68% and 25.938% field $\mathbb{F}_p$-operations for CSIDH-512, CSIDH-1024, and CSIDH-1792, respectively. These results are somewhat more encouraging than the ones reported in [4], where speedups of about 1% and 8% were reported for a non constant-time implementation of CSIDH-512 and CSIDH-1024.

5.2 Experiments playing the B-SIDH

To the best of our knowledge, we present in this section the first implementation of the B-SIDH protocol, which was designed to be a constant-time one. As in the case of CSIDH, we report here the required number of $\mathbb{F}_p$ arithmetic operations. Similarly to CSIDH, the B-SIDH implementation provided in this work, allows to perform experiments with any prime field of p elements such that $p \equiv 3 \bmod 4$. The main contribution provided in this subsection corresponds to a comparison of B-SIDH instantiations using the primes **B-SIDHp253**, **B-SIDHp255**,**B-SIDHp247**,**B-SIDHp237** and **B-SIDHp257**, as described in Appendix C.

All the above primes were chosen considering the following features: i) $p \equiv 3 \bmod 4$, ii) the isogeny degrees are as small as it was possible to find, and iii) $2^{210} < N, M$. Our Python3-code implementation uses the degree-4 isogeny construction and evaluation formulae given in [11]. The corresponding experimental results for the key generation and

Configuration	Group action evaluation	M	S	a	Cost	Saving (%)
tvelu	OAYT-style	0.641	0.172	0.610	0.813	—
	MCR-style	0.835	0.231	0.785	1.066	
	dummy-free	1.246	0.323	1.161	1.569	
svelu	OAYT-style	0.656	0.178	0.988	0.834	−2.583
	MCR-style	0.852	0.219	1.295	1.071	−0.469
	dummy-free	1.257	0.324	1.888	1.581	−0.765
hvelu	OAYT-style	0.624	0.165	0.893	0.789	2.952
	MCR-style	0.805	0.204	1.164	1.009	5.347
	dummy-free	1.198	0.301	1.696	1.499	4.461

Table 1: Number of field operation for the constant-time CSIDH-512 group action evaluation. Counts are given in millions of operations, averaged over 1024 random experiments. For computing the Cost column, it is assumed that $\mathbf{M} = \mathbf{S}$ and all addition counts are ignored. Last column labeled **Saving** corresponds to $\left(1 - \frac{\mathbf{Cost}}{\text{baseline}}\right) \times 100$ and baseline equals to *tvelu* configuration.

Configuration	Group action evaluation	M	S	a	Cost	Saving (%)
tvelu	OAYT-style	0.630	0.152	0.576	0.782	—
	MCR-style	0.775	0.190	0.695	0.965	
	dummy-free	1.152	0.259	1.012	1.411	
svelu	OAYT-style	0.566	0.138	0.963	0.704	9.974
	MCR-style	0.702	0.152	1.191	0.854	11.503
	dummy-free	1.046	0.230	1.746	1.276	9.568
hvelu	OAYT-style	0.552	0.133	0.924	0.685	12.404
	MCR-style	0.687	0.146	1.148	0.833	13.679
	dummy-free	1.027	0.221	1.679	1.248	11.552

Table 2: Number of field operation for the constant-time CSIDH-1024 group action evaluation. Counts are given in millions of operations, averaged over 1024 random experiments. For computing the Cost column, it is assumed that $\mathbf{M} = \mathbf{S}$ and all addition counts are ignored. Last column labeled **Saving** corresponds to $\left(1 - \frac{\mathbf{Cost}}{\text{baseline}}\right) \times 100$ and baseline equals to *tvelu* configuration.

secret sharing phases are presented in Table 4 and Table 5, respectively. It can be seen that significant savings ranging from 24% up to 76% were obtained by B-SIDH combined with $\sqrt{}$élu with respect to the same implementation of this protocol using traditional Vélu's formulae.

Notice that the best results were obtained when using the **B-SIDHp253** configuration, which seems to be faster than any CSIDH instantiation, mostly due to its small 256-bit field.

5.3 Discussion

Table 6 presents the clock cycle counts for several isogeny-based protocols recently reported in the literature. Rather than providing a direct comparison, the main purpose of including this table here is that of providing a perspective of the relative timing costs of several emblematic implementations of isogeny-based key-exchange primitives.

Configuration	Group action evaluation	M	S	a	Cost	Saving (%)
tvelu	OAYT-style	1.385	0.263	1.137	1.648	—
	MCR-style	1.041	0.239	0.911	1.280	
	dummy-free	1.557	0.327	1.336	1.884	
svelu	OAYT-style	1.063	0.187	2.073	1.250	24.150
	MCR-style	0.807	0.154	1.550	0.961	24.922
	dummy-free	1.233	0.247	2.314	1.480	21.444
hvelu	OAYT-style	1.060	0.185	2.061	1.245	24.454
	MCR-style	0.797	0.151	1.522	0.948	25.938
	dummy-free	1.220	0.241	2.272	1.461	22.452

Table 3: Number of field operation for the constant-time CSIDH-1792 group action evaluation. Counts are given in millions of operations, averaged over 1024 random experiments. For computing the Cost column, it is assumed that $\mathbf{M} = \mathbf{S}$ and all addition counts are ignored. Last column labeled **Saving** corresponds to $\left(1 - \frac{\mathbf{Cost}}{\text{baseline}}\right) \times 100$ and baseline equals to *tvelu* configuration.

Configuration		Alice's side			Bob's side		
		M	a	Saving (%)	M	a	Saving (%)
tvelu	**B-SIDHp253**	4.229	8.731	—	3.444	7.107	—
	B-SIDHp255	4.254	8.774		2.900	5.984	
	B-SIDHp247	0.910	1.881		2.295	4.735	
	B-SIDHp237	0.077	0.164		10.449	21.532	
	B-SIDHp257	4.281	8.828		0.303	0.630	
svelu	**B-SIDHp253**	1.176	4.403	72.192	0.972	3.750	71.777
	B-SIDHp255	1.225	4.664	71.204	0.879	3.252	69.690
	B-SIDHp247	0.452	1.492	50.330	0.997	3.423	56.558
	B-SIDHp237	0.106	0.243	−37.663	2.772	10.684	73.471
	B-SIDHp257	1.332	4.933	68.886	0.230	0.665	24.092
hvelu	**B-SIDHp253**	1.158	4.355	72.618	0.953	3.699	72.329
	B-SIDHp255	1.223	4.659	71.251	0.867	3.221	70.103
	B-SIDHp247	0.442	1.461	51.429	0.995	3.420	56.645
	B-SIDHp237	0.077	0.164	00.000	2.770	10.676	73.490
	B-SIDHp257	1.321	4.905	69.143	0.217	0.633	28.383

Table 4: Number of base field operation in $\mathbb{F}_p$ for the public key generation phase of BSIDH. Counts are given in millions of operations. Columns labeled **Saving** correspond to $\left(1 - \frac{\mathbf{Cost}}{\text{baseline}}\right) \times 100$ and baseline equals to *tvelu* configuration.

Clearly, √élu has a dramatic impact on the performance of B-SIDH, so much so that one can claim confidently that B-SIDH outperforms any instantiation of CSIDH. For example, using the B-SIDH configuration presented in example 2 of [12], Alice and Bob will require about 1.620×2^{20} and 1.343×2^{20} base field multiplications in $\mathbb{F}_p$, where p is a 256-bit prime, respectively. In particular, making the conservative assumption that a 256-bit field multiplication takes 40 clock cycles, then a key exchange using B-SIDH would cost about 118.520×2^{20} clock cycles. On the other hand, the fastest CISDH-512 group action evaluation (see [19, 10]) takes about 230×2^{20} clock cycles. Therefore, a key

Configuration		Alice's side			Bob's side		
		M	**a**	**Saving (%)**	**M**	**a**	**Saving (%)**
tvelu	**B-SIDHp253**	1.831	3.936	—	1.529	3.277	—
	B-SIDHp255	1.931	4.127		1.305	2.795	
	B-SIDHp247	0.434	0.928		1.113	2.372	
	B-SIDHp237	0.053	0.115		4.872	10.377	
	B-SIDHp257	1.963	4.190		0.156	0.336	
svelu	**B-SIDHp253**	0.472	1.769	74.222	0.400	1.546	73.839
	B-SIDHp255	0.505	1.945	73.847	0.370	1.357	71.648
	B-SIDHp247	0.208	0.668	52.074	0.450	1.543	59.569
	B-SIDHp237	0.068	0.157	−28.302	1.184	4.590	75.698
	B-SIDHp257	0.562	2.094	71.370	0.116	0.327	25.641
hvelu	**B-SIDHp253**	0.462	1.741	74.768	0.390	1.517	74.493
	B-SIDHp255	0.505	1.943	73.847	0.362	1.338	72.261
	B-SIDHp247	0.203	0.653	53.226	0.449	1.541	59.659
	B-SIDHp237	0.053	0.115	00.000	1.183	4.585	75.718
	B-SIDHp257	0.555	2.077	71.727	0.108	0.306	30.769

Table 5: Number of base field operation in $\mathbb{F}_p$ for the secret sharing phase of BSIDH. Counts are given in millions of operations. Columns labeled **Saving** correspond to $\left(1 - \frac{\textbf{Cost}}{\text{baseline}}\right) \times 100$ and baseline equals to *tvelu* configuration.

Implementation	**Protocol Instantiation**	**Mcycles**
SIKE [2]	SIKEp434	22
Castryck *et al.* [8]	CSIDH-512 unprotected	4×155
Bernstein *et al.* [4]	CSIDH-512 unprotected	4×153
	CSIDH-1024 unprotected	4×760
Cervantes-Vázquez *et al.* [9]	CSIDH-512 MCR-style	4×339
	CSIDH-512 OAYT-style	4×238
Hutchinson *et al.* [19]	CSIDH-512 OAYT-style	4×229
Chi-Domínguez*et al.* [10]	CSIDH-512 MCR-style	4×298
	CSIDH-512 OAYT-style	4×230
*This work (**estimated**)*	CSIDH-512 MCR-style	4×282
	CSIDH-512 OAYT-style	4×223
	B-SIDH-p253	119

Table 6: Skylake Clock cycle timings for a key exchange protocol for different instantiations of the SIDH, CSIDH, and B-SIDH protocols.

exchange using CSIDH would take about 920×2^{20} clock cycles (considering four group action evaluations). This implies that B-SIDH is expected to be about 8x faster than the fastest CSIDH-512 C-code implementation.

Costello proposed in [12] that B-SIDH could be useful for key-exchange scenarios executed in the context of a client-server session. Typically, one could expect that the client has much more constrained computational resources than the server. In the case

that the prime B-SIDHp237 is chosen for performing a B-SIDH key exchange, Alice and Bob would require about 0.13×2^{20} and 3.953×2^{20} base field multiplications in $\mathbb{F}_p$. Assuming once again that that a 256-bit field multiplication takes 40 clock cycles, then a key exchange using B-SIDH would cost about 5.20×2^{20} and 158.12×2^{20} clock cycles for Alice and Bob, respectively. For comparison, a SIKEp434 key exchange costs about 10.73×2^{20} and 12.04×2^{20} clock cycles for Alice and Bob, respectively. Hence, Alice (the client) will benefit with a B-SIDHp237 computation that is about twice as fast as the one required in SIKEp434. This will come at the price that Bob's computation (the server) would become thirteen times more expensive. On the other hand, the B-SIDHp237 key sizes are noticeably smaller than the ones required in SIKEp434. This feature is especially valuable for highly constrained client devices.

We stress that the quantum security level offered by the CSIDH instantiations reported in this work have been recently call into question in [28, 6].

In terms of security, the B-SIDH instantiations reported in this paper should achieve the same classical and quantum security level than a SIDH instantiations using the SIKEp434 prime. However, B-SIDH is susceptible to the active attack described in [18]. To offer protection against this kind of attacks, B-SIDH should incorporate a key encapsulation mechanism such as the one included in [2]. Providing this protection will imply an extra overhead for B-SIDH, which was not considered in this paper.

6 Conclusions

A concrete analysis of $\sqrt{\text{élu}}$ introduced in [4] was presented in this paper. From our analysis we conclude that for most practical scenarios, the best approach for performing the polynomial products associated to $\sqrt{\text{élu}}$, is achieved by Karatsuba polynomial multiplications. The main practical consequence of this observation is that computing degree-ℓ isogenies with $\sqrt{\text{élu}}$ has a *concrete* computational complexity essentially proportional to $b^{\log_2(3)}$, where $b = \sqrt{\ell}$.

We introduced several algorithmic tricks that permit to save multiplications when performing the polynomial products involving the computation of the resultants included in Algorithm 2-Algorithm 3. The combination of these improvements allows us to construct and evaluate degree-ℓ isogenies with a slightly lesser number of arithmetic operations than the ones employed in [4].

We applied $\sqrt{\text{élu}}$ and optimal strategies to several instantiations of the CSIDH and B-SIDH protocols, producing the very first constant-time implementation of the latter protocol for a selection of primes taken from [12, 4].

Our future work includes C constant-time single-core and multi-core implementations of the two protocol instantiations studied in this work. We would also like to study more efficient selections of the sets $\mathcal{I}, \mathcal{J}$ and $\mathcal{K}$ as defined in §4.1, which could yield more economical computations of $\sqrt{\text{élu}}$.

Acknowledgements. This work was partially done while the third author was visiting the University of Waterloo. The third author received partial funds from the Mexican

Science council CONACyT project 313572. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 804476). This work was partially supported by the Spanish Ministerio de Ciencia, Innovación y Universidades, under the reference MTM2017-83271-R.

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 - 25th International Conference*, volume 11349 of *Lecture Notes in Computer Science*, pages 322–343. Springer, 2018.

[2] Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, David Jao, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Geovandro Pereira, Joost Renes, Vladimir Soukharev, and David Urbanik. Supersingular isogeny key encapsulation. second round candidate of the nist's post-quantum cryptography standardization process, 2017. Available at: https://sike.org/.

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

[4] Daniel J. Bernstein, Luca De Feo, Antonin Leroux, and Benjamin Smith. Faster computation of isogenies of large prime degree. *IACR Cryptol. ePrint Arch.*, 2020:341, 2020.

[5] Daniel J. Bernstein, Tanja Lange, Chloe Martindale, and Lorenz Panny. Quantum circuits for the CSIDH: optimizing quantum evaluation of isogenies. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 409–441. Springer, 2019.

[6] Xavier Bonnetain and André Schrottenloher. Quantum security analysis of CSIDH. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020, Proceedings, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 493–522. Springer, 2020.

[7] J. W. S. Cassels. *Lectures on Elliptic Curves*. London Mathematical Society Student Texts, 24 edition, 2016.

[8] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: an efficient post-quantum commutative group action. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018, Part III*, volume 11274 of *Lecture Notes in Computer Science*, pages 395–427. Springer, 2018.

[9] Daniel Cervantes-Vázquez, Mathilde Chenu, Jesús-Javier Chi-Domínguez, Luca De Feo, Francisco Rodríguez-Henríquez, and Benjamin Smith. Stronger and faster side-channel protections for CSIDH. In Peter Schwabe and Nicolas Thériault, editors, *Progress in Cryptology - LATINCRYPT 2019*, volume 11774 of *Lecture Notes in Computer Science*, pages 173–193. Springer, 2019.

[10] Jesús-Javier Chi-Domínguez and Francisco Rodríguez-Henríquez. Optimal strategies for CSIDH. *IACR Cryptol. ePrint Arch.*, 2020:417, 2020.

[11] Deirdre Connolly. Code for sidh key exchange with optional public key compression. Github, April 2017. available at: `https://github.com/dconnolly/msr-sidh/tree/master/SIDH-Magma`.

[12] Craig Costello. B-SIDH: supersingular isogeny diffie-hellman using twisted torsion. *IACR Cryptol. ePrint Arch.*, 2019:1145, 2019.

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

[14] 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*, pages 572–601, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[15] Jean-Marc Couveignes. Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291, 2006. `http://eprint.iacr.org/2006/291`.

[16] Luca De Feo, David Jao, and Jérôme Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *J. Math. Cryptol.*, 8(3):209–247, 2014.

[17] Luca De Feo, Jean Kieffer, and Benjamin Smith. Towards practical key exchange from ordinary isogeny graphs. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018, Part III*, volume 11274 of *Lecture Notes in Computer Science*, pages 365–394. Springer, 2018.

[18] Steven D. Galbraith, Christophe Petit, Barak Shani, and Yan Bo Ti. On the security of supersingular isogeny cryptosystems. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 63–91, 2016.

[19] Aaron Hutchinson, Jason T. LeGrow, Brian Koziel, and Reza Azarderakhsh. Further optimizations of CSIDH: A systematic approach to efficient strategies, permutations, and bound vectors. *IACR Cryptol. ePrint Arch.*, 2019:1121, 2019.

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

[21] David 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`.

[22] Michael Meyer, Fabio Campos, and Steffen Reith. On lions and elligators: An efficient constant-time implementation of CSIDH. In Jintai Ding and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 10th International Conference*, volume 11505 of *Lecture Notes in Computer Science*, pages 307–325. Springer, 2019.

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

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

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

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

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

[28] Chris Peikert. He gives c-sieves on the CSIDH. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - Proceedings, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 463–492. Springer, 2020.

[29] Joost Renes. Computing isogenies between montgomery curves using the action of (0, 0). In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*, volume 10786 of *Lecture Notes in Computer Science*, pages 229–247. Springer, 2018.

[30] Alexander Rostovtsev and Anton Stolbunov. Public-key cryptosystem based on isogenies. *IACR Cryptology ePrint Archive*, 2006:145, 2006.

[31] Anton Stolbunov. Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves. *Adv. in Math. of Comm.*, 4(2):215–235, 2010.

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

A Algorithms

Algorithm 4 Simplified constant-time CSIDH class group action for supersingular curves over $\mathbb{F}_p$, where $p = 4\prod_{i=1}^{n} \ell_i - 1$. The ideals $\mathfrak{l}_i = (\ell_i, \pi - 1)$, where π maps to the p-th power Frobenius endomorphism on each curve. This algorithm computes exactly m isogenies for each ideal $\mathfrak{l}_i$ (Adapted from [10]).

Require: A supersingular curve E_A over $\mathbb{F}_p$, and an exponent vector $(e_1, \ldots, e_n)$ with each $e_i \in [0, m]$), m a positive number.
Ensure: $E_B = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E_A$.

$E_0 \leftarrow E$ // Initializing to the base curve
// Outer loop: Each ℓ_i is processed m times
for $i \leftarrow 1$ to m **do**
  $T \leftarrow$ `GetFullTorsionPoint`(E_0) // $T \in E_n[\pi - 1]$
  $T \leftarrow [4]T$ // Now $T \in E_n\left[\prod_i \ell_i\right]$
  // Inner loop: processing each prime factor $\ell_i | (p+1)$
  for $j \leftarrow 0$ to $(n-1)$ **do**
    $G_j \leftarrow T$
    for $k \leftarrow 1$ to $(n-1-j)$ **do**
      $G_j \leftarrow [\ell_k]G_j$
    end for
    if $e_{n-j} \neq 0$ **then**
      $\langle G_j \rangle \leftarrow$ `KPS`(G_j)
      $E_{(j+1) \bmod n} \leftarrow$ `xISOG`$(E_j, \ell_{n-j}, \langle G_j \rangle)$
      $T \leftarrow$ `xEVAL`$(T, \langle G_j \rangle)$
      $e_{n-j} \leftarrow e_{n-j} - 1$
    else
      $\langle G_j \rangle \leftarrow$ `KPS`(G_j)
      `xISOG`$(E_j, \ell_{n-j}, \langle G_j \rangle)$ // Dummy operations
      $T \leftarrow [\ell_{n-j}]T$
      $E_{j+1 \bmod n} \leftarrow E_j$
    end if
  end for
end for
return E_0

B Computational cost of computing resultants via remainder trees

In this section we focused on the computational cost associated to a resultant computation via remainder trees. Resultants are required by the √élu procedures `xISOG` and `xEVAL`.

Formally, each one of the two resultants required by Algorithm 2 and Algorithm 3, corresponds to the computation of $\mathtt{Res}_Z(f(Z), g(Z))$ such that $f, g \in \mathbb{F}_q[Z]$, $\deg f = b' \approx b$ and $\deg g = 2b$. Our goal in this Appendix is that of deriving the cost of the resultant computation in terms of b. For the sake of simplicity, let us assume $\deg f = b$.

It is important to highlight that the modular polynomial reduction required at each node in the remainder tree, can be performed via reciprocal computations (for more

Algorithm 5 Large composite degree isogeny construction

Require: a supersingular Montgomery curve $E/\mathbb{F}_{p^2} : By^2 = x^3 + Ax^2 + x$, a kernel point generator R on $E/\mathbb{F}_{p^2}$ of order $L \coloneqq {\ell_1}^{e_1} \cdot {\ell_2}^{e_2} \cdots {\ell_n}^{e_n}$, and a strategy S

Ensure: the degree-L isogenous curve $E/\langle R \rangle$

Set L' as in Equation 2 // S must be determined by L'
$ramifications \leftarrow [R]$ // list of points to be evaluated
$moves \leftarrow [0]$; $k \leftarrow 0$
$e \leftarrow \#L'$ // e must be equal to $\#S + 1$
// Outer loop: Each ℓ_i is processed e_i times
for $i \coloneqq 0$ to $\#S - 1$ **do**
  $prev \leftarrow sum(moves)$
  // Inner loop: computing the kernel point generator
  while $prev < (e - 1 - i)$ **do**
    $moves.append(S_k)$
    $V \leftarrow$ last element of $ramifications$
    for $j \coloneqq prev$ to $prev + S_k$ **do**
      $V \leftarrow [L'_j]V$
    end for
    $ramifications.append(V)$ // New point to be evaluated
    $prev \leftarrow prev + S_k$
    $k \leftarrow k + 1$
  end while
  $G \leftarrow$ last element of $ramifications$
  $\langle G \rangle \leftarrow \texttt{KPS}(G)$
  $E \leftarrow \texttt{xISOG}(E, \ell_{e-1-i}, \langle G \rangle)$
  // Inner loop: evaluating points
  for $j \coloneqq 0$ to $\#moves - 1$ **do**
    $ramifications_j \leftarrow \texttt{xEVAL}(ramifications_j, \langle G \rangle)$
  end for
  $moves.pop()$
  $ramifications.pop()$
end for
$G \leftarrow$ the unique element of $ramifications$
$\langle G \rangle \leftarrow \texttt{KPS}(G)$
$E \leftarrow \texttt{xISOG}(E, \ell_0, \langle G \rangle)$
return E

details see [3, p. 27, §17]). For example, the modular polynomial reduction $g \bmod f$ requires two degree-b polynomial multiplications modulo x^b, one constant multiplication by a degree-b polynomial, and the reciprocal computation modulo x^b (that is, $1/f \bmod x^b$). In turn, the cost of a reciprocal computation modulo x^b can be estimated by the expenses associated to two degree-$(b/2)$ polynomial multiplications modulo $x^{b/2}$, one constant multiplication by a degree-$(b/2)$ polynomial, and another reciprocal, but this time modulo $x^{(b/2)}$. The above implies that a reciprocal modulo x^b should be computed recursively. Its associated running time complexity equation is given as,

$$T(b) = T\left(\frac{b}{2}\right) + 2t\left(\frac{b}{2}\right) + \frac{b}{2},$$

where $t(b)$ denotes the polynomial multiplication cost of two degree-b polynomials modulo x^b. Now, assuming that a Karatsuba polynomial multiplication is used, it follows that

$$\begin{aligned} T(b) &\approx T\left(\frac{b}{2}\right) + 2\left(\frac{b}{2}\right)^{\log_2(3)} + \frac{b}{2} = T\left(\frac{b}{2}\right) + \frac{2}{3}b^{\log_2(3)} + \frac{b}{2} \\ &= \sum_{i=0}^{\log_2(b)} \left(\frac{2}{3}\left(\frac{b}{2^i}\right)^{\log_2(3)} + \frac{b}{2^{i+1}}\right) \\ &= \left(\frac{2}{3}b^{\log_2(3)}\right) \sum_{i=0}^{\log_2(b)} \frac{1}{3^i} + \left(\frac{b}{2}\right) \sum_{i=0}^{\log_2(b)} \frac{1}{2^i} \\ &= \left(1 - \frac{1}{3^{\log_2(b)+1}}\right) b^{\log_2(3)} + \left(1 - \frac{1}{2^{\log_2(b)+1}}\right) b\,. \end{aligned}$$

Hence, the polynomial reduction $g \bmod f$ is expected to have a running time of $\left(\left(3 - \frac{1}{3^{\log_2(b)+1}}\right) b^{\log_2(3)} + \left(2 - \frac{1}{2^{\log_2(b)+1}}\right) b\right)$ field multiplications.

Now, the remainder tree of f and g is constructed going from its root all the way to its leaves. To do this, at the i-th level of the remainder tree 2^i modular reductions of the form $g \bmod f$ such that $\deg f \approx \frac{b}{2^i}$ and $\deg g \approx 2 \deg f$, must be performed. Their combined cost is given as,

$$\begin{aligned} R(b,i) &\coloneqq \left(2^i\right)\left(\left(3 - \frac{1}{3^{\log_2(b/2^i)+1}}\right)\left(\frac{b}{2^i}\right)^{\log_2(3)} + \left(2 - \frac{1}{2^{\log_2(b/2^i)+1}}\right)\left(\frac{b}{2^i}\right)\right) \\ &= \left(2^i\right)\left(\left(3 - \frac{1}{3^{\log_2(b)-i+1}}\right)\left(\frac{b^{\log_2(3)}}{3^i}\right) + \left(2 - \frac{1}{2^{\log_2(b)-i+1}}\right)\left(\frac{b}{2^i}\right)\right) \\ &= 3b^{\log_2(3)}\left(\left(\frac{2}{3}\right)^i - \left(\frac{2^i}{3^{\log_2(b)}}\right)\right) + \left(2 - \frac{2^i}{2^{\log_2(b)+1}}\right) b\,. \end{aligned}$$

Furthermore, the cost of the remainder tree construction can be done with about $R(b) \coloneqq \sum_{i=0}^{\log_2(b)} R(b,i)$ field multiplications. In particular,

$$R(b) = 9b^{\log_2(3)}\left(1 - \left(\frac{2}{3}\right)^{\log_2(b)+1} - \left(\frac{2^{\log_2(b)+1}}{3^{\log_2(b)+1}}\right)\right) + \left(2\log_2(b) - \frac{2^{\log_2(b)+1}}{2^{\log_2(b)+1}}\right) b$$
$$= 9b^{\log_2(3)}\left(1 - 2\left(\frac{2}{3}\right)^{\log_2(b)+1}\right) + \big(2\log_2(b) - 1\big)b\,.$$

Finally, once the remainder tree has been constructed, the next step is to multiply all its leaves, which has an extra cost of b field multiplications, and produces that the Resultant $\texttt{Res}_Z(f(Z), g(Z))$ computation requires a total of

$$\left(9b^{\log_2(3)}\left(1 - 2\left(\frac{2}{3}\right)^{\log_2(b)+1}\right) + 2b\log_2(b)\right)\mathbf{M}.$$

On the other hand, the polynomials required in the root of the remainder tree can be obtained via product trees at a cost of $\left(\left(1 - \frac{1}{3^{\log_2(b)+1}}\right) b^{\log_2(3)}\right)$ field multiplications.

C B-SIDH primes

1. Example 2. of [12, section 5.2], we named it as **B-SIDHp253**:

$$\begin{aligned} p &= \texttt{0x1935BECE108DC6C0AAD0712181BB1A414E6A8AAA6B510FC29826} \\ &\quad \texttt{190FE7EDA80F}, \\ M &= 4^2 \cdot 3 \cdot 7^{16} \cdot 17^9 \cdot 31^8 \cdot 311 \cdot 571 \cdot 1321 \cdot 5119 \cdot 6011 \cdot 14207 \cdot 28477 \\ &\quad \cdot 76667, \text{ and} \\ N &= 11^{18} \cdot 19 \cdot 23^{13} \cdot 47 \cdot 79 \cdot 83 \cdot 89 \cdot 151 \cdot 3347 \cdot 17449 \cdot 33461 \cdot 51193. \end{aligned}$$

2. Example 3. of [12, section 5.2], we named it as **B-SIDHp255**:

$$\begin{aligned} p &= \texttt{0x76042798BBFB78AEBD02490BD2635DEC131ABFFFFFFFFFFFFFFF} \\ &\quad \texttt{FFFFFFFFFFFF}, \\ M &= 4^{55} \cdot 5 \cdot 7^2 \cdot 67 \cdot 223 \cdot 4229 \cdot 9787 \cdot 13399 \cdot 21521 \cdot 32257 \cdot 47353, \\ &\quad \text{and} \\ N &= 3^{34} \cdot 11 \cdot 17 \cdot 19^2 \cdot 29 \cdot 37 \cdot 53^2 \cdot 97 \cdot 107 \cdot 109 \cdot 131 \cdot 137 \cdot 197 \cdot 199 \\ &\quad \cdot 227 \cdot 251 \cdot 5519 \cdot 9091 \cdot 33997 \cdot 38201. \end{aligned}$$

3. Example 5. of [12, section 5.3], we named it as **B-SIDHp247**:

$$
\begin{aligned}
p &= \texttt{0x46B27D6FAE96ED4A639E045B7D2C3CA33F476892ADAFF87B9B6E} \\
&\quad \texttt{AE5EE1FFFF}, \\
M &= \left(4^2 \cdot 5^2 \cdot 7 \cdot 23 \cdot 79 \cdot 107 \cdot 307 \cdot 2129\right)^4 \cdot 7901^2, \text{ and} \\
N &= 3 \cdot 11 \cdot 17 \cdot 241 \cdot 349 \cdot 421 \cdot 613 \cdot 983 \cdot 1327 \cdot 1667 \cdot 2969 \cdot 3769 \cdot \\
&\quad 4481 \cdot 4649 \cdot 4801 \cdot 4877 \cdot 5527 \cdot 6673 \cdot 7103 \cdot 7537 \cdot 7621.
\end{aligned}
$$

4. Example 6. of [12, section 5.3], we named it as **B-SIDHp237**:

$$
\begin{aligned}
p &= \texttt{0x1B40F93CE52A207249237A4FF37425A798E914A74949FA343E8E} \\
&\quad \texttt{A487FFFF}, \\
M &= 4^3 \cdot \left(4 \cdot 3^4 \cdot 17 \cdot 19 \cdot 31 \cdot 37 \cdot 53^2\right)^6, \text{ and} \\
N &= 7 \cdot 13 \cdot 43 \cdot 73 \cdot 103 \cdot 269 \cdot 439 \cdot 881 \cdot 883 \cdot 1321 \cdot 5479 \cdot 9181 \cdot \\
&\quad 12541 \cdot 15803 \cdot 20161 \cdot 24043 \cdot 34843 \cdot 48437 \cdot 62753 \cdot 72577.
\end{aligned}
$$

5. Lucky proposal of [4, appendix A], we named it as **B-SIDHp257**:

$$
\begin{aligned}
p &= \texttt{0x1E409D8D53CF3BEB65B5F41FB53B25EBEAF37761CD8BA9966841} \\
&\quad \texttt{50A40FFFFFFFF}, \\
M &= 4^{16} \cdot 5^{21} \cdot 7 \cdot 11 \cdot 163 \cdot 1181 \cdot 2389 \cdot 5233 \cdot 8353 \cdot 10139 \cdot 11939 \cdot \\
&\quad 22003 \cdot 25391 \cdot 41843, \text{ and} \\
N &= 3^{56} \cdot 31 \cdot 43 \cdot 59 \cdot 271 \cdot 311 \cdot 353 \cdot 461 \cdot 593 \cdot 607 \cdot 647 \cdot 691 \cdot 743 \cdot \\
&\quad 769 \cdot 877 \cdot 1549.
\end{aligned}
$$