Journal of Cryptographic Engineering (2023) 13:89–106
https://doi.org/10.1007/s13389-022-00293-y

REGULAR PAPER

Karatsuba-based square-root Vélu's formulas applied to two isogeny-based protocols

Gora Adj[1] · Jesús-Javier Chi-Domínguez[2] · Francisco Rodríguez-Henríquez[2,3]

Received: 16 December 2021 / Accepted: 21 June 2022 / Published online: 20 July 2022

Abstract

At a combined computational cost of about 6ℓ field operations, Vélu's formulas are used to construct and evaluate degree-ℓ isogenies in the vast majority of isogeny-based cryptographic schemes. By adapting to Vélu's formulas a baby-step giant-step approach, Bernstein, De Feo, Leroux, and Smith presented a procedure that can compute isogeny operations at a reduced cost of just $\tilde{O}(\sqrt{\ell})$ field operations. In this paper, we present a concrete computational analysis of these novel procedures along with several algorithmic tricks that helped us to further decrease its computational cost. We also report an optimized Python3-code implementation of several instantiations of two isogeny-based key-exchange protocols, namely, CSIDH and B-SIDH. Our software library uses a combination of the modified Vélu's formulas and an adaptation of the optimal strategies commonly used in the SIDH/SIKE protocols to produce significant speedups. 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, we present the first optimized implementation of B-SIDH ever reported in the open literature.

Keywords Isogenies · Square-root Vélu formulas · Karatsuba-based complexity analysis · CSIDH · B-SIDH

1 Introduction

Isogeny-based cryptography was independently introduced in 2006 by Couveignes [16], Rostovtsev and Stolbunov in [32,34]. 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, is briefly summarized below.

Operating with supersingular elliptic curves defined over the finite field $\mathbb{F}_{p^2}$, with p a prime, the Supersingular Isogeny-based Diffie-Hellman key exchange protocol (SIDH) was presented by Jao and De Feo in [21] (see also [17]). In 2017, the Supersingular Isogeny Key Encapsulation (SIKE) protocol, an SIDH variant, was submitted to the NIST post-quantum cryptography standardization project [2]. On July 2020, NIST announced that SIKE passed to the round 3 of this contest as an alternate 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 a prime field $\mathbb{F}_p$, CSIDH is a significantly faster version of the Couveignes-Rostovtsev-Stolbunov scheme variant as it was presented in [18].

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. 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. 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.

✉ Jesús-Javier Chi-Domínguez
jesus.dominguez@tii.ae

Gora Adj
gora.adj@gmail.com

Francisco Rodríguez-Henríquez
francisco.rodriguez@tii.ae

1 Departament de Matemàtica, Universitat de Lleida, Lleida, Spain

2 Cryptography Research Centre, Technology Innovation Institute, Abu Dhabi, United Arab Emirates

3 Computer Science Department, CINVESTAV-IPN, Mexico City, Mexico

More recently in 2021, Banegas *et al.* proposed in [3] a more efficient approach for computing constant-time CSIDH that they named CTIDH. In CTIDH the authors employ an economical key space mechanism (which was adapted from an idea of bounding the 1-norm of CSIDH secret key vector [28]), along with an approach for processing primes in batches [24].

In general, performing isogeny constructions and evaluations are the most expensive computational tasks of any isogeny-based protocol. This is especially true for CSIDH and B-SIDH, where [exceedingly] large odd prime degree-ℓ isogenies come into play.

For decades now, Vélu's formulas (cf. [22, Sect. 2.4] and [35, Theorem 12.16]) have been widely used to construct and evaluate degree-ℓ isogenies. Using several elliptic curve and isogeny arithmetic optimization tricks reported in the last few years [9,14,27], the construction and evaluation of degree-ℓ isogenies via Vélu's formulas can be obtained at a computational cost of roughly 6ℓ field multiplications (see a detailed discussion in Sect. 2).

Bernstein, De Feo, Leroux and Smith presented in [5] 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 main polynomial product embedded in the isogeny computations, can be effectively accelerated via a baby-step giant-step approach [5, Algorithm 2]. Due to its square root complexity reduction (up to polylogarithm factors), in the remainder of this paper, we will refer to this variant of Vélu's formulas, as $\sqrt{\text{é}}$lu formulas or simply $\sqrt{\text{é}}$lu.

As we will see in this paper, and as it was already hinted in [5], $\sqrt{\text{é}}$lu has a noticeable impact on the performance of CSIDH, and even more so 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 [5, 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. 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 [5], along with the high performance achieved by its companion software library, the authors did not provide a practical cost analysis of their approach, but rather, they focus their attention on its asymptotical analysis. Moreover, their $\sqrt{\text{é}}$lu implementation reported a rather modest 1% and 8% speedup over the traditional Vélu's formulas when applied to the non constant-time CSIDH-512 and CSIDH-1024 instantiations, respectively. Furthermore, the authors of [5] left open the problem of assessing the practical impact of $\sqrt{\text{é}}$lu on CSIDH and B-SIDH constant-time implementations.

Our Contributions. We present a concrete computational analysis of $\sqrt{\text{é}}$lu. 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 $\sqrt{\text{é}}$lu has a concrete computational cost dominated by a $b^{\log_2(3)}$ factor, where $b \approx \frac{\sqrt{(\ell-1)}}{2}$. We also present several tricks that permit to save multiplications when performing products involving the polynomials E_{J_0} and E_{J_1} as defined in Sect. 4. We additionally exploit the fact that the polynomials E_{J_0} and E_{J_1} are the reciprocal of each other. These simple but effective observations help us to construct and evaluate a degree-587 isogeny using only $2180M \approx 3.701(\ell + 2)$. This is about 5.3% cheaper than the same computation announced in [5]. This improvement also pushes to $\ell = 89$ the threshold where computing degree-ℓ isogenies with $\sqrt{\text{é}}$lu becomes more effective than traditional Vélu.[2]

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

1. We report the first constant-time implementation of the protocol B-SIDH introduced in [13]. Using the framework of [11], optimal strategies à la SIDH are applied to B-SIDH while also taking advantage of $\sqrt{\text{é}}$lu. The experimental results for B-SIDH show a saving of up to 75% compared with an implementation of this protocol using traditional Vélu.
2. We used the framework presented in [11] to apply optimal strategies to CSIDH, while exploiting $\sqrt{\text{é}}$lu. This allows us to present the first application of $\sqrt{\text{é}}$lu to constant-time implementations of the CSIDH-512, CSIDH-1024, and CSIDH-1792 instantiations. A comparison with respect to CSIDH using traditional Vélu, 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. In Sect. 4.3, we show that the number of field multiplications required for computing degree-ℓ isogenies using $\sqrt{\text{é}}$lu with Karatsuba polynomial multiplication has concrete computational cost closer to $O(b^{\log_2(3)})$.

Outline. The remainder of this paper is organized as follows. In Sect. 2, we give a description of traditional Vélu's

[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.

[2] Recently, Banegas et al. [3, Sect. 7.2] reported an even lower count for this computation. The authors construct and evaluate a degree-587 isogeny at a cost of just 2108 multiplications, which is 3.3% cheaper than the cost reported in this work.

formulas. We include also a compact description of the B-SIDH and CSIDH protocols. In Sect. 3, we briefly discuss the application of optimal strategies to CSIDH and B-SIDH. In Sect. 4, we present an explicit description of $\sqrt{\text{é}}$lu main building blocks KPS, xEVAL, and xISOG. In addition, we discuss several $\sqrt{\text{é}}$lu algorithmic improvements in Sect. 4.2. We report the experimental results obtained from our software library in Sect. 5, first in Sect. 5.1 for CSIDH and then in Sect. 5.2 for B-SIDH. Finally, our concluding remarks are drawn in Sect. 6.

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

2 Background

Most if not all of the fastest isogeny-based constant-time protocol implementations have adopted for their schemes Montgomery and twisted Edwards curve models. A Montgomery curve [26] 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 ℓ be an odd prime number. Given an order-ℓ point $P \in E(\mathbb{F}_q)$, the construction of a degree-ℓ isogeny $\phi : E \mapsto E'$ of kernel $G = \langle P \rangle$ and its evaluation at a point $Q \in E(\mathbb{F}_q) \backslash G$ consist 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 this paper, we will refer to these two tasks as isogeny construction and isogeny evaluation computations, respectively.

Vélu's formulas (see [22, Sect. 2.4] and [35, Theorem12.16]), have been generally used to construct and evaluate degree-ℓ isogenies by performing three main building blocks known as, 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 codomain curve E'. Also, using KPS as a building block, xEVAL calculates the x-coordinate of the image point $\phi(Q) \in E'$.

After applying a number of elliptic curve arithmetic tricks [9,14,27], the computational expenses of KPS, xISOG and xEVALhave been found to be about 3ℓ, ℓ 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 Sect. 4, we give a detailed discussion of how the $\sqrt{\text{é}}$lu approach of [5] drastically reduces the timing costs of traditional Vélu's formulas.[3]

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

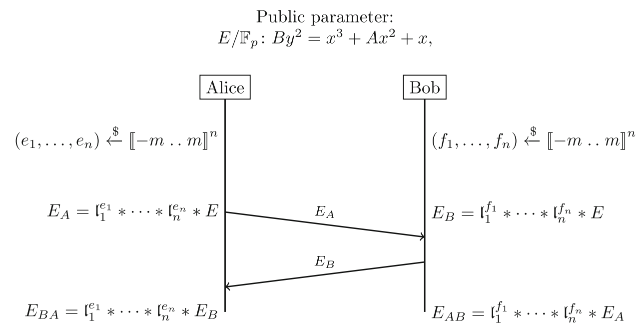


Fig. 1 CSIDH key-exchange protocol

2.1 Overviewing the CSIDH

Here, we give a simplified description of CSIDH. For more technical details, the interested reader is referred to [8,9,24, 29].

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.

In CSIDH, we compute the action of small prime ideals, which happen to be factors of $p+1$. We have that the principal ideal $(\ell_i) \subset \mathbb{Z}[\sqrt{-p}]$ splits into two primes, namely $\mathfrak{l}_i = (\ell_i, \pi - 1)$ and $\bar{\mathfrak{l}}_i = (\ell_i, \pi + 1)$, where π is the Frobenius endomorphism. From the fact that $\bar{\mathfrak{l}}_i \mathfrak{l}_i = (\ell_i)$ is principal, we obtain CSIDH commutative property, $\bar{\mathfrak{l}}_i * (\mathfrak{l}_i * E) =$

[3] This speedup is achieved as a time-memory trade-off: an optimized implementation of $\sqrt{\text{é}}$lu requires much more memory than traditional Vélu.

$\mathfrak{l}_i * (\bar{\mathfrak{l}}_i * E) = E$, for all $E/\mathbb{F}_p$ with endomorphism ring $\mathrm{End}(E) \cong \mathbb{Z}[\sqrt{-p}]$.

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

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

Taking advantage of the commutative property of the group action, we can implement the protocol shown in Fig. 1, 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,

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

The computational cost of the group action described in 4 of Sect. A.1, is dominated by the calculation of n degree-$\ell_i^{e_i}$ 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 [6, Sect. 8] (see also [20]) that this multiplication-based procedure could possibly be improved by adapting to CSIDH, the SIDH optimal strategy approach introduced by De Feo, Jao and Plût in [17]. We briefly discuss about the role of optimal strategies for large instances of CSIDH in Sect. 3, where the framework presented in [11] was adopted.

2.2 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 Fig. 2.[5]

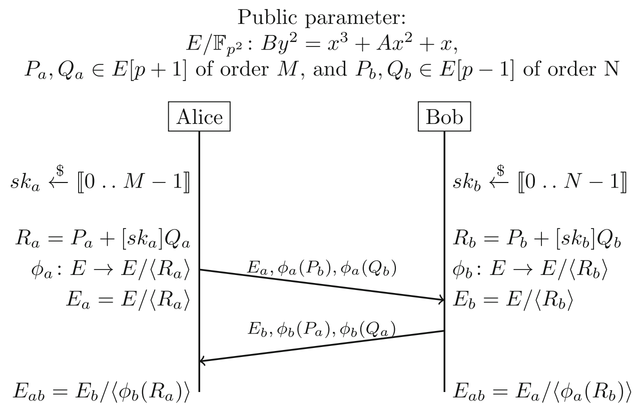


Fig. 2 B-SIDH protocol for a prime p such that $M|(p+1)$ and $N|(p-1)$.

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 $E/\mathbb{F}_{p^2}$. For efficiency reasons, the prime p is chosen so that both, $p+1$ and $p-1$ are M- and N-smooth, respectively. In other words, $M|(p+1)$ and $N|(p-1)$. 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 $\#E_t(\mathbb{F}_{p^2}) = (p-1)^2$. Note 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, Sect. 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 $\pi : (x, y) \mapsto (x^p, y^p)$ 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 Fig. 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 $E/\mathbb{F}_{p^2}: By^2 = x^3 + Ax^2 + x$ 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

[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 Fig. 2, they are completely analogous to the ones corresponding to SIDH, a protocol that is carefully discussed in many papers such as [1,15,17].

- 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.

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

3 Optimal strategies for the CSIDH and the B-SIDH

In [17], 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 [2,11] for concrete algorithms).

In the context of SIDH, optimal strategies tend to balance the number of isogeny evaluations and scalar multiplications to $O(e \log(e))$. In the case of CSIDH, optimal strategies are expected to be largely multiplicative, *i.e.*, optimal strategies will tend to favor the computation of more scalar multiplications over isogeny evaluations. This is due to the fact that these operations are cheaper than large prime degree-ℓ isogeny evaluations.

Let $\mathcal{L} = [\ell_1, \ell_2, \ldots, \ell_n]$ 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. Here, we adopt the framework presented in [11], where the authors heuristically assumed that an arrangement of the set $\mathcal{L}$ from the smallest to the largest ℓ_i, is close to the global optimal. For this fixed ordering, the authors of [11] reported a procedure that finds an optimal strategy with cubic complexity with respect to n.

Optimal strategies can also be used to improve the performance of B-SIDH, although in this case, we can see the resulting strategies as a hybrid between SIDH and CSIDH. On the one hand, B-SIDH follows the same SIDH protocol flow. On the other hand, B-SIDH must construct/evaluate several isogenies whose degrees are powers of large odd primes, as in CSIDH.

Let us assume that we need to construct a degree-L isogeny with $L = {\ell_1}^{e_1} \cdot {\ell_2}^{e_2} \cdots {\ell_n}^{e_n}$, and let us write

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

Then, in order to efficiently execute either the key generation or the secret sharing main phases of B-SIDH, we must find an optimal strategy for the setting $\mathcal{L}'$ as described in Algoritm 5 of "Appendix A.1".

Notice that any B-SIDH strategy can be encoded as is customary in SIDH and CSIDH, *i.e.*, by a list of $e-1$ positive integers where $e = \sum_{i=1}^{n} e_i$. Moreover, any such strategy can be evaluated by executing the dynamic-programming procedure shown in Algorithm 5.

4 New Vélu's formulas

In this section we present a more detailed discussion of the $\sqrt{}$élu algorithms and their application to isogeny-based cryptography. We give several algorithmic tricks that slightly improve the performance of $\sqrt{}$élu as it was presented in [5].

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 [5] (see also [14], [25] and [27]), the following formulas allow to accomplish this task,

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

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

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 $\sqrt{}$élu 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-ℓ 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 [5],

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

where,

$$F_0(Z, X) = Z^2 - 2XZ + X^2; \quad (3)$$
$$F_1(Z, X) = -2(XZ^2 + (X^2 + 2AX + 1)Z + X);$$
$$F_2(Z, X) = X^2Z^2 - 2XZ + 1.$$

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

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

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

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

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

4.1 Construction and evaluation of odd degree isogenies

As in Sect. 2, we consider the three building blocks `KPS`, `xISOG`, `xEVAL`, where `KPS` consists of computing 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 line with the traditional approach, one could use the `KPS` procedure of traditional Vélu for computing the x coordinates of ($\#S = (\ell - 1)/2$) points in the kernel G. This will cost about 3ℓ field multiplications. More efficiently, $\sqrt{\text{é}}$lu 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. We call $(\mathcal{I}, \mathcal{J})$ an *index system for* S and write $\mathcal{I} \pm \mathcal{J}$ for $(\mathcal{I} + \mathcal{J}) \cup (\mathcal{I} - \mathcal{J})$. 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 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 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** $\mathcal{I}, \mathcal{J}, \mathcal{K}$

Let us recall that for achieving an efficient computation of `xISOG` and `xEVAL` , $\sqrt{\text{é}}$lu requires the biquadratic polynomials of Eq. 3, which implies the computation of resultants of the form $\mathrm{Res}_Z(f(Z), g(Z))$, for two polynomials $f, g \in \mathbb{F}_q[Z]$.

We are now ready to present in Algorithms 2–3 the computation of `xISOG` and `xEVAL`, respectively. Deriving the resultants in Algorithms 2–3 may turn out to be a cumbersome task if it is not carried out in an elaborated way. For polynomials $f = a\prod_{0 \le i < n}(Z - x_i)$ and g in $\mathbb{F}_q[Z]$, their resultant $\mathrm{Res}(f, g) = a^n \prod_{0 \le i < n} g(x_i)$ 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 $g \bmod (Z - x_i), 0 \le i < n$, followed by their product.

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 3, since these linear polynomials depend only on the kernel $\langle P \rangle$. In other words, the four resultants in Algorithms 2–3 show no dependencies among them and therefore, they can be computed concurrently by a $\sqrt{\text{é}}$lu parallel implementation.

Notice that the single most recurrent high level operation of Algorithms 2–3, is the polynomial multiplication on the ring $\mathbb{F}_q[X]$. Thus, as in [5], it is essential that we utilize fast tailor-made polynomial multiplication algorithms. These customized algorithms are useful because for several modular computations, only a segment of the polynomial product is actually needed.

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 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 $\langle P \rangle$.

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

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 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 $\langle P \rangle$.

$h_I \leftarrow \prod_{x_i \in \mathcal{I}} (Z - x_i)) \in \mathbb{F}_q[Z]$
$E_{0,J} \leftarrow \prod_{x_j \in \mathcal{J}} \left(\frac{F_0(Z,x_j)}{\alpha^2} + \frac{F_1(Z,x_j)}{\alpha} + F_2(Z, x_j)\right) \in \mathbb{F}_q[Z]$
$E_{1,J} \leftarrow \prod_{x_j \in \mathcal{J}} \left(F_0(Z, x_j)\alpha^2 + F_1(Z, x_j)\alpha + F_2(Z, x_j)\right) \in \mathbb{F}_q[Z]$
$R_0 \leftarrow \mathtt{Res}_Z(h_I, E_{0,J}) \in \mathbb{F}_q$
$R_1 \leftarrow \mathtt{Res}_Z(h_I, E_{1,J}) \in \mathbb{F}_q$
$M_0 \leftarrow \prod_{x_k \in \mathcal{K}} (1/\alpha - x_k) \in \mathbb{F}_q$
$M_1 \leftarrow \prod_{x_k \in \mathcal{K}} (\alpha - x_k) \in \mathbb{F}_q$
return $(M_0 R_0)^2/(M_1 R_1)^2$

The resultant $\mathtt{Res}_Z(f(Z), g(Z))$ 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 $O(n \log_2(n))$ computational complexity (see [4, p. 7, Sect. 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. For example, according to the implementation of [5], $\ell = 587$ requires polynomials of degree $\#\mathcal{I} = 16$ and $2 \times \#\mathcal{J} = 18$ (in the B-SIDH case this translates to $\#\mathcal{I}, \#\mathcal{J} \leq 150$). It can be easily verified that Karatsuba polynomial multiplication becomes a more efficient choice than the Schönage-FFT approach (for a comprehensive analysis of these design options, see "Appendix A.2").

4.2 Implementation speedups

In this section we report several algorithmic techniques that are exploited in our implementation to obtain some modest, but noticeably savings over [5]. Our first refinement affects xEVAL, and arises from the special shape of the biquadratic polynomials F_0, F_1, F_2. Considering either variable, one can see that F_1 is symmetric and F_0 is symmetric to F_2[6], that is, $F_1 = 1/Z^2\,F_1(1/Z, X)$ and $F_2 = 1/Z^2\,F_0(1/Z, X)$ by, for example, considering the first variable. 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 $E_{0,j}$ required in xEVAL. Let us write $x_j = X_j/Z_j$. 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

$$a = C(xZ_j - zX_j)^2;$$
$$2b = \left[C(X^2 + Z^2)\right](-4X_jZ_j) - \left[2(X_j^2 + Z_j^2)\right]\left(2[C(XZ)]\right) + \left(2[A'(XZ)]\right)(-4X_jZ_j);$$
$$c = C(xX_j - zZ_j)^2.$$

The three equations above can be implemented (with the help of some extra pre-computations required in xISOG) at a cost of 7M + 3S + 12a field operations. This cost represents a saving with respect to the implementation of [5], which requires 11M + 2S + 13a field operations. Assuming M= **S**, this implies that our proposed formulas save 3 field multiplications per polynomial $E_{0,j}$, $0 \leq j < \#\mathcal{J}$.

Let us now illustrate the improvements just described applied to the example $\ell = 587$. Let us recall that in the implementation of [5], 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.

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

4.3 A concrete complexity analysis

In this section, we present the computational cost associated to the combined evaluation of the `KPS`, `xISOG`, and `xEVAL` procedures.[7]

Let $b = \lfloor \frac{\sqrt{\ell-1}}{2} \rfloor$ as given in Step 1 of Algorithm 1. Note that `KPS` (see Algorithm 1), can be performed at a cost of about $4b$ differential point additions (assuming $\#\mathcal{I} \approx \#\mathcal{J} \approx b, \#\mathcal{K} \approx 2b$), which implies an expense of at most $(24b)M$ field multiplications.

Observe also that the computation of the polynomial $h_I(Z)$ required at Step 1 of both, `xISOG` (2) and `xEVAL` (3) procedures, can be shared and thus must be computed only once. One interesting observation of [5], is that the computation of the polynomials $E_{0,J}$ and $E_{1,J}$ in `xISOG` (see Steps 2–3 of `xISOG` 2), can be performed at a cost of only one product tree procedure. Furthermore, as it was already discussed in Sect. 4.2, this same trick can also be applied to `xEVAL`, *i.e.*, Steps 2–3 of `xEVAL` 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 $(3b)M$ and $(10b)M$ 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 $\mathrm{Res}_Z(f(Z), g(Z))$ such that $f, g \in \mathbb{F}_q[Z]$, $\deg f = b' \approx b$ and $\deg g = 2b$. We give in "Appendix A.3", a detailed description of the cost of computing such a resultant in terms of b. This calculation is performed by computing the product of the remainder tree leaves. In "Appendix A.3", it is shown that the complexity in terms of field operations associated to the computation of a resultant as described in Sect. 4.2 is given as,

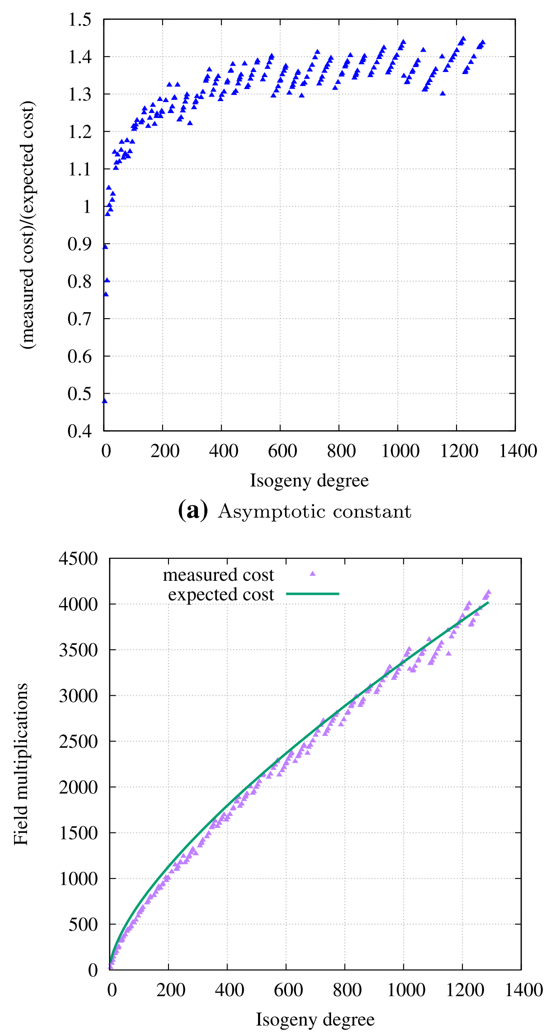
$$R(b) = \left(3b^{\log_2(3)} + b\log_2(b) - \frac{5}{3}b + \frac{5}{6}\right). \tag{4}$$

The constants M_0 and M_1 in Steps 6-7 of `xISOG` and `xEVAL`, have a cost of $(4b)M$ and $(8b)M$ field operations, respectively. Lastly, the computations of the coefficient d of `xISOG` and the output of `xEVAL` require about $(6\log_2(b) + 22)$ multiplications. All in all and invoking Eq. 4, the evaluation of `KPS`, `xISOG`, and `xEVAL` procedures have a combined cost of approximately,

$$\begin{aligned}\mathrm{Cost}(b) &= 4\left(3b^{\log_2(3)} + b\log_2(b) - \frac{5}{3}b + \frac{5}{6}\right) \\ &\quad + \left(b^{\log_2(3)} - \frac{2}{3}b\right) + 2\left(3b^{\log_2(3)} - 2b\right) \\ &\quad + 49b + 6\log_2(b) + 22 \\ &= 19b^{\log_2(3)} + 4b\log_2(b) + \frac{113}{3}b + 6\log_2(b) + \frac{76}{3}. \end{aligned} \tag{5}$$

Recall that $\mathcal{K}$ corresponds to the missing kernel computation part not included in the remainder trees, whose size is upper bounded by $2b$ elements. The precise size of $\mathcal{K}$ depends on the choice of the prime ℓ, which in turn determines the value of b. Since our model does not account for the cost associated with $\mathcal{K}$, it is just natural to expect a slight experimental discrepancy as shown in Fig. 3a, where we present the ratio between experimental and expected isogeny costs. For the sake of simplicity, we did not delve into further refinements of our model.

However, to quantify the correctness of the cost predicted by Eq. 5, we performed the following experiment.



(a) Asymptotic constant

(b) Running time

Fig. 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 [11]). All computational costs are given in $\mathbb{F}_p$-multiplications. The expected running time corresponds to $1.4 \times \mathrm{Cost}(b)$. Additionally, $b \approx \frac{\sqrt{(\ell-1)}}{2}$

[7] In the sequel, $\sqrt{\text{é}}$lu computational costs are derived assuming a projective coordinate system and $M = S$.

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 [11]. Figure 3 shows an accurate correlation between the theoretical cost of Eq. 5 and the experimental results obtained from our Python3 library software, where we can see that the number of required field multiplications is at most 1.4 times larger than the expected value predicted by our analysis above.

Recall that the derivation of the expected cost of Eq. 5 (See "Appendix A.3"), is driven by the assumption that $M = S$, 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 $\mathbb{F}_{q=p^2}$, it holds that $M_{\mathbb{F}_q} = 3M_{\mathbb{F}_p}$ and $S_{\mathbb{F}_q} = 2M_{\mathbb{F}_p}$, and thus $S_{\mathbb{F}_q} = \frac{2}{3}M_{\mathbb{F}_q}$. However, as an upper bound (for the B-SIDH case), we can assume $M_{\mathbb{F}_q} = 3M_{\mathbb{F}_p}$ and $M_{\mathbb{F}_q} = S_{\mathbb{F}_q}$, which gives an expected running-time of $3 \times \text{Cost}(b)$ $\mathbb{F}_p$-multiplications.

A memory analysis of $\sqrt{\text{é}}$lu reveals that less than $4b$ points, equivalent to $8b$ field elements, are computed and stored in `KPS`. The computation of the trees determined by the polynomial h_I in Step 1 of `xISOG` and `xEVAL`, requires the storage of no more than $3b\log_2 b$ field elements.[8] All in all, $\sqrt{\text{é}}$lu memory cost is of about $8b + 3b\log_2 b$ field elements.

By performing a quick inspection of Algorithms 1–3, one can see 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 Algorithms 2–3 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 $\sqrt{\text{é}}$lu, the expected savings are substantial.

5 Experiments and discussion

In this section, we introduce the Python3-code constant-time library **sibc** (Supersingular Isogeny-Based Cryptographic constructions), dedicated to isogeny-based primitives. The **sibc** library aims to easily compare, test, and run SIDH-based primitives such as SIDH, SIKE, CSIDH, and BSIDH.

We remark that our CSIDH and B-SIDH implementations make extensive usage of the $\sqrt{\text{é}}$lu formulas introduced in [5], boosted with the computational tricks presented in Sect. 4. Furthermore, we also exploit the optimal strategy framework presented in [11], which helps us to maximize the performance of both protocols.

[8] For this computation two remainder trees are constructed, requiring the storage of $2b\log_2 b$ field elements. In addition, the recursivity procedure to build the trees may require storing in the heap space another $b\log_2 b$ field elements.

In summary, our Python3-code software allows us to readily 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 adding, multiplying, and squaring field elements. 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, we use standard arithmetic tricks over $\mathbb{F}_{p^2}$, to perform the multiplication and squaring at a cost of $3M + 5a$ and $2M + 3a$ 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 formulas [14,31] and élu:

- Using tradicional Vélu (labeled as *tvelu*);
- Using $\sqrt{\text{é}}$lu (labeled as *svelu*);
- Using a hybrid between traditional Vélu and $\sqrt{\text{é}}$lu (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. OAYT-style [29]: two torsion point with dummy isogeny constructions;
2. MCR-style [24]: one torsion point with dummy isogeny constructions;
3. Dummy-free style [9]: two torsion point without dummy isogeny constructions.

Our software supports performing experiments with any prime field of $p = 2^e \cdot \left(\prod_{i=1}^{n} \ell_i\right) - 1$ elements, for any $e \geq 1$. Our experiments were focused on the CSIDH-512 prime proposed in [8], the CSIDH-1024 prime proposed in [5], and the CSIDH-1792 prime proposed in [11]. We stress that the quantum security level offered by the CSIDH instantiations reported in this work have been recently call into question in [7,10,30].

The required number of field operations for those CSIDH variants are reported in Tables 1, 2, and 3. In addition, each table presents a comparison between the results of this work and the ones presented in [11]. Moreover, for each configuration we adopted optimal strategies and suitable bound vectors according to [11, Sects. 3.4, 4.4 and 4.5].

When comparing with respect to CSIDH constant-time implementations using traditional Vélu's formulas, 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 [5], where speedups of about 1% and 8% were reported for a non constant-time implementation of CSIDH-512 and CSIDH-1024.

CTIDH, a fast constant-time implementation of CSIDH (*cf.* with Table 6), was recently reported in [3]. CTIDH combines an efficient key space mechanism along with a *Matryoshka*-structure-like for processing several degree-ℓ isogenies in batches [23]. CTIDH's key space is upper bounded by B_i, the 1-norm of the i-th batch. Security wise, it is critical that all the isogenies associated to a given batch are calculated in constant-time. To assure this, all the isogenies in a batch are computed at the cost of the maximal degree of this batch. This can only be achieved by adding a significant number of dummy operations.

It appears clear that $\sqrt{\text{é}}$lu plays a significant role for achieving CTIDH's impressive computational performance. Unfortunately, the authors of [3] did not include a version of CTIDH using traditional Vélu's formulas. This experiment would have allowed to precisely quantify the speedup contributed by $\sqrt{\text{é}}$lu alone in the CTIDH performance.

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

Table 1 Number of field operation for the constant-time CSIDH-512 group action evaluation

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

Counts are given in millions of operations, averaged over 1024 random experiments. For computing the Cost column, it is assumed that **M** = **S** and all addition counts are ignored. Last column labeled Saving corresponds to $\left(1 - \frac{\textbf{Cost}}{\text{baseline}}\right) \times 100$ and baseline equals to *tvelu* configuration

Table 2 Number of field operation for the constant-time CSIDH-1024 group action evaluation

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

Counts are given in millions of operations, averaged over 1024 random experiments. For computing the Cost column, it is assumed that **M** = **S** and all addition counts are ignored. Last column labeled Saving corresponds to $\left(1 - \frac{\textbf{Cost}}{\text{baseline}}\right) \times 100$ and baseline equals to *tvelu* configuration

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

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

For computing the Cost column, it is assumed that **M = S** and all addition counts are ignored. Last column labeled Saving corresponds to $\left(1 - \frac{\textbf{Cost}}{\text{baseline}}\right) \times 100$ and baseline equals to *tvelu* configuration

Table 4 Number of base field operation in $\mathbb{F}_p$ for the public key generation phase of BSIDH

Configuration		Alice's side			Bob's side		
		M	a	Saving (%)	M	a	Saving (%)
tvelu	B-SIDHp253	3.835	8.077	–	3.129	6.584	–
	B-SIDHp255	3.874	8.144	–	2.639	5.552	–
	B-SIDHp247	0.836	1.760	–	2.101	4.413	–
	B-SIDHp237	0.079	0.169	–	9.523	19.988	–
	B-SIDHp257	3.901	8.197	–	0.287	0.607	–
svelu	B-SIDHp253	0.951	3.469	75.212	0.788	2.950	74.805
	B-SIDHp255	0.995	3.693	74.328	0.716	2.585	72.881
	B-SIDHp247	0.380	1.225	54.577	0.827	2.774	60.644
	B-SIDHp237	0.104	0.243	− 32.701	2.236	8.480	76.523
	B-SIDHp257	1.084	3.916	72.206	0.205	0.575	28.447
hvelu	B-SIDHp253	0.935	3.427	75.623	0.772	2.907	75.316
	B-SIDHp255	0.994	3.689	74.356	0.705	2.558	73.277
	B-SIDHp247	0.372	1.200	55.538	0.826	2.771	60.701
	B-SIDHp237	0.081	0.176	-2.867	2.234	8.473	76.544
	B-SIDHp257	1.074	3.892	72.469	0.194	0.548	32.403

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

B-SIDHp253, B-SIDHp255,B-SIDHp247,B-SIDHp237 and B-SIDHp257, which are specified in "Appendix A.4".

All the above primes were chosen considering the following features: (i) $p \equiv 3 \bmod 4$, (ii) the parameters $M | (p + 1)$ and $N | (p - 1)$ are as smooth as it was possible to find, and (iii) $2^{210} < N, M$. Our Python3-code implementation uses the degree-4 isogeny construction and evaluation formulas given in [12]. Additionally, the key generation does not perform `xISOG` calls, which are expensive for large primes, it reconstructs the A-coefficient by using the three points pushed under the isogeny being computed (that is, we implement a projective version of `get_A`() procedure). The corresponding experimental results for the key generation and secret sharing phases are presented in Tables 4 and 5, respectively. It can be seen that significant savings ranging from 24% up to 76% were obtained by B-SIDH combined with $\sqrt{\text{é}}$lu with respect to the same implementation of this protocol using traditional Vélu's formulas.

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

Table 5 Number of base field operation in $\mathbb{F}_p$ for the secret sharing phase of BSIDH

Configuration		Alice's side			Bob's side		
		M	a	Saving (%)	M	a	Saving (%)
tvelu	B-SIDHp253	1.838	3.948	–	1.534	3.285	–
	B-SIDHp255	1.937	4.138	–	1.311	2.804	–
	B-SIDHp247	0.439	0.938	–	1.118	2.379	–
	B-SIDHp237	0.058	0.124	–	4.877	10.384	–
	B-SIDHp257	1.969	4.202	–	0.164	0.351	–
svelu	B-SIDHp253	0.480	1.785	73.882	0.408	1.563	73.392
	B-SIDHp255	0.513	1.961	73.521	0.378	1.374	71.198
	B-SIDHp247	0.215	0.684	50.982	0.458	1.558	59.058
	B-SIDHp237	0.076	0.175	− 30.377	1.191	4.605	75.576
	B-SIDHp257	0.569	2.111	71.078	0.124	0.343	24.502
hvelu	B-SIDHp253	0.470	1.757	74.449	0.397	1.533	74.101
	B-SIDHp255	0.512	1.959	73.548	0.370	1.355	71.734
	B-SIDHp247	0.210	0.668	52.121	0.457	1.556	59.132
	B-SIDHp237	0.060	0.131	− 3.878	1.190	4.601	75.603
	B-SIDHp257	0.562	2.093	71.431	0.116	0.324	29.029

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

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

Implementation	Protocol Instantiation	Mcycles
SIKE [2]	SIKEp434	22
Castryck et al. [8]	CSIDH-512 unprotected	4 × 155
Bernstein et al. [5]	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. [20]	CSIDH-512 OAYT-style	4 × 229
Chi-Domínguez et al. [11]	CSIDH-512 MCR-style	4 × 298
	CSIDH-512 OAYT-style	4 × 230
Banegas et al. [3]	CTIDH-512	4 × 126
	CTIDH-1024	4 × 470
This work (estimated)	CSIDH-512 MCR-style	4 × 282
	CSIDH-512 OAYT-style	4 × 223
	B-SIDH-p253	119

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.

Clearly, $\sqrt{\text{é}}$lu 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 [13], 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 [11,20]) takes about 230×2^{20} clock cycles. Therefore, a key 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 as a possible application for B-SIDH, key exchange protocols executed in the context of a client-

Table 7 Number of base field operation in $\mathbb{F}_p$ of both SIKE and B-SIKE (B-SIDH with KEM) protocol.

Algorithm	Security	KeyGen		Encaps		Decaps	
		M	a	M	a	M	a
SIKEp434	NIST LEVEL 1	0.043	0.096	0.074	0.159	0.077	0.170
SIKEp503	NIST LEVEL 2	0.051	0.114	0.087	0.188	0.092	0.200
SIKEp610	NIST LEVEL 3	0.063	0.140	0.118	0.254	0.118	0.258
SIKEp751	NIST LEVEL 5	0.080	0.177	0.136	0.292	0.143	0.312
B-SIKEp253	NIST LEVEL 1	0.772	2.907	1.404	5.185	1.332	4.960

Counts are given in millions of operations. `Encaps` and `Decaps` denote the key encapsulation and decapsulation, respectively

server session [13]. Typically, one could expect that the client has much more constrained computational resources than the server. In the case that we choose the prime B-SIDHp237 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 $\mathbb{F}_p$. Once again, assuming 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.

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 [19]. To offer protection against this kind of attacks, B-SIDH should incorporate a key encapsulation mechanism (KEM) such as the one included in [2]. Essentially, a B-SIDH augmented version with a key encapsulation mechanism (B-SIKE), inherits the same SIKE protocol flow: (i) `KeyGen` performs one degree-M isogeny, ii) `Encaps` computes two ephemeral degree-N isogenies, and iii) `Decaps` executes one degree-M isogeny and one ephemeral degree-N isogeny.

We illustrate the impact of a KEM in B-SIDH, comparing in Table 7 the associated timings of SIKE and B-SIKE instantiations with similar security level. In particular, we focus on our best B-SIDH instantiation: B-SIDHp253 with KEM (B-SIKEp253).

Once again, assuming that a 253-bit field multiplication takes 40 clock cycles, then a B-SIKEp253 instantiation would cost $(0.772 + 1.404 + 1.332) \times 40.0 \approx 140.32$ Millions of clock cycles. This is still faster than any CSIDH-512 instantiation, and also faster than CTIDH-512 [3], which is about twice as fast as CSIDH-512) [9]

6 Conclusions

In this paper, we presented a concrete analysis of the $\sqrt{\text{é}}$lu procedure introduced in [5]. From our analysis, we conclude that for most practical scenarios, the best approach for performing the polynomial products associated to $\sqrt{\text{é}}$lu, is Karatsuba polynomial multiplication. The main concrete consequence of this observation is that computing degree-ℓ isogenies with $\sqrt{\text{é}}$lu has a *practical* computational complexity essentially proportional to $b^{\log_2(3)}$, where $b \approx \frac{\sqrt{(\ell-1)}}{2}$.

We introduced several algorithmic tricks that permit to save multiplications when performing the polynomial products involving the computation of the resultants included in 2–3. 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 [5].

We applied $\sqrt{\text{é}}$lu 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 [5,13].

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 Sect. 4.1, which could yield more economical computations of $\sqrt{\text{é}}$lu.

Acknowledgements We thank the anonymous reviewers for their comments to improve the quality of the paper and Amalia Pizarro and Odalis Ortega for pointing a missed factor in the product tree cost analysis.

Funding This project started when J. Chi-Domínguez was a postdoctoral researcher at Tampere University, and initially received funding from the European Commission through the ERC Starting Grant 804476. It received funds from the Mexican Science council CONACyT project 313572, while F. Rodríguez-Henríquez was visiting the Univer-

[9] Our python-code implementation of SIDH is based on the SIDH specifications [2].

sity of Waterloo. Additionally, this work was partially supported by the Spanish Ministerio de Ciencia, Innovación y Universidades, under the reference MTM2017-83271-R.

Availability of data and material Run-time data is not available

Code availability Our software library is freely available at https://github.com/JJChiDguez/sibc.

Declarations

Conflicts of interest The authors have no conflicts of interest to declare.

A Appendix

A.1 Algorithms

Algorithm 4 Simplified constant-time CSIDH class group action for supersingular curves over $\mathbb{F}_p$ $p = 4\prod_{i=1}^{n} \ell_i - 1$. The ideals $\mathfrak{l}_i = (\ell_i, \pi - 1)$, where π maps to the p-th power Frobenius morphism. This algorithm computes exactly m isogenies for each ideal $\mathfrak{l}_i$ (Adapted from [11]).

Require: A supersingular curve E_A over $\mathbb{F}_p$, an integer vector $(e_1, \ldots, e_n) \in [\![0 \,..\, m]\!]^n$, $m > 0$.
Ensure: $E_B = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E_A$.

```
E_0 ← E   {Initializing to the base curve}
{Outer loop: Each ℓ_i is processed m times}
for i ← 1 to m do
    T ← GetFullTorsionPoint(E_0)  {T ∈ E_n[π − 1] }
    T ← [4]T {Now T ∈ E_n [∏_i ℓ_i]}
    {Inner loop: processing each prime factor ℓ_i|(p + 1)}
    for j ← 0 to (n − 1) do
        G_j ← T
        for k ← 1 to (n − 1 − j) do
            G_j ← [ℓ_k]G_j
        end for
        if e_{n−j} ≠ 0 then
            ⟨G_j⟩ ← KPS(G_j)
            E_{(j+1) mod n} ← xISOG(E_j, ℓ_{n−j}, ⟨G_j⟩)
            T ← xEVAL(T, ⟨G_j⟩)
            e_{n−j} ← e_{n−j} − 1
        else
            ⟨G_j⟩ ← KPS(G_j)
            xISOG(E_j, ℓ_{n−j}, ⟨G_j⟩) {Dummy operations}
            T ← [ℓ_{n−j}]T
            E_{j+1 mod n} ← E_j
        end if
    end for
end for
return E_0
```

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 $L = \ell_1{}^{e_1} \cdot \ell_2{}^{e_2} \cdots \ell_n{}^{e_n}$, 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 ← [R] {list of points to be evaluated}
moves ← [0]; k ← 0
e ← #L′ {e must be equal to #S + 1}
{Outer loop: Each ℓ_i is processed e_i times}
for i ← 0 to #S − 1 do
    prev ← sum(moves)
    {Inner loop: computing the kernel point generator}
    while prev < (e − 1 − i) do
        moves.append(S_k)
        V ← last element of ramifications
        for j ← prev to prev + S_k do
            V ← [L′_j]V
        end for
        ramifications.append(V) {New point to be evaluated}
        prev ← prev + S_k; k ← k + 1
    end while
    G ← last element of ramifications
    ⟨G⟩ ← KPS(G)
    E ← xISOG(E, ℓ_{e−1−i}, ⟨G⟩)
    {Inner loop: evaluating points}
    for j ← 0 to #moves − 1 do
        ramifications_j ← xEVAL(ramifications_j, ⟨G⟩)
    end for
    moves.pop(); ramifications.pop()
end for
G ← the unique element of ramifications
⟨G⟩ ← KPS(G)
E ← xISOG(E, ℓ_0, ⟨G⟩)
return E
```

A.2 Schönage-FFT vs Karatsuba

Karatsuba multiplication is a well-known and complete tool for multiplying polynomials of degree n over a commutative ring at the subquadratic cost of $O(n^{\log_2 3})$. However, an asymtotically faster family of algorithms based on the fast Fourier transform (FFT) exists. In this section, we consider Schönage's algorithm [33] blended with the FFT multiplication, as described in [4], and give an accurate estimate of the running time of this algorithm in order to make practical comparatives with Karatsuba multiplication.

Let A be a commutative ring where 2 in invertible. For $n > 1$ a power of 2, c a square in A and $\zeta \in A$ a square root of -1, let f, g be two polynomials in $A[x]/(x^n + c)$. To multiply f and g, one can split the problem into two smaller ones by reducing f, g to $f_-, g_- \in A[x]/(x^{n/2} - \zeta c^{1/2})$ and to $f_+, g_+ \in A[x]/(x^{n/2} + \zeta c^{1/2})$g. Then, the products $f_- g_-$, $f_+ g_+$ are computed, and subsequently embedded into $A[x]/(x^n + c)$ wherein $(f_- g_- + f_+ g_+)$ and $(f_- g_- - f_+ g_+)$ are calculated to finally recover $2fg$.

Note that when c is an nth root in A, which in addition contains an nth root of -1, then the above procedure can be applied recursively to compute the product nfg at a cost of k multiplications in A and $\frac{3}{2} n \log_2(n)$ easy multiplications in A by constants. This is essentially the FFT multiplication.

Suppose now that A does not contain an nth root of -1, with $n = 2^s > 8$, then Schönage's method can be employed to multiply $f = \sum_{0 \le i < n} f_i$ and $g = \sum_{0 \le i < n} g_i$ in $A[x]/(x^n + 1)$. First, define $n_1 = 2^{s_1}$, with $s_1 = \lfloor s/2 \rfloor$, $B = A[x]/(x^{n_1} + 1)$, and consider the ring $B[y]/(y^{2n/n_1} + 1)$. The goal here is to reduce the computation of fg into one multiplication in $B[y]/(y^{2n/n_1} + 1)$. Note that $x^{n_1^2/2n}$ is a $(2n/n_1)$th root of -1 in B, and hence the FFT can be used to multiply polynomials in $B[y]/(y^{2n/n_1} + 1)$. We start by sending f, g to $F, G \in A[x, y]/(y^{2n/n_1} + 1)$, respectively, where

$$F = \sum_{0\le j<\frac{2n}{n_1}} \sum_{0\le i<\frac{n_1}{2}} f_{i+\frac{n}{2}j} x^i y^j \text{ and}$$
$$G = \sum_{0\le j<\frac{2n}{n_1}} \sum_{0\le i<\frac{n_1}{2}} g_{i+\frac{n}{2}j} x^i y^j,$$

are such that $\phi(F) = f$ and $\phi(G) = g$, the map $\phi : A[x, y]/(y^{2n/n_1} + 1) \to A[x]/(x^n + 1)$ being the $A[x]$-algebra morphism that sends y to x^{n_1}. Thus, since F and G have x-degree $< n_1/2$, their product is computed in $B[y]/(y^{2n/n_1} + 1)$, and then passed through ϕ to recover $(2n/n_1)fg$.

To estimate the cost of this computation, notice that transforming f, g to F, G and $(2n/n_1)FG$ to $(2n/n_1)fg$ requires no multiplications in A. Moreover, when computing $(2n/n_1)FG$ in $B[y]/(y^{2n/n_1} + 1)$ using the FFT, the multiplications by constants can be ignored since these will be just multiplications by powers of x in B. Therefore, the cost of multiplying polynomials in $A[x]/(x^n+1)$ boils down to the $2n/m$ multiplications in B arising from the FFT application. Now, since $B = A[x]/(x^{n_1} + 1)$, the above strategy can be applied recursively until reaching multiplications in $A[x]/(x^8 + 1)$, where more conventional methods can be used. Hence, the total cost of multiplying two polynomials in $A[x]/(x^n + 1)$ will be

$$C(n) = \frac{2n}{n_1} \times \frac{2n_1}{n_2} \times \cdots \times \frac{2n_{k-1}}{n_k} \times C_8 = 2^k \frac{n}{n_k} C_8,$$

where $n_i = 2^{s_i}$, with $s_i = \lfloor s_{i-1}/2 \rfloor$ for $i \in \{2, \ldots, k\}$, k is such that $n_k = 8$, and C_8 is the cost of multiplying two polynomials in $A[x]/(x^8 + 1)$. An easy analysis then shows that $k = \lceil \log_2(s-1)\rceil - 1 = \lceil \log_2(\log_2(n)-1)\rceil - 1$. Thus, we have

$$C(n) = \frac{C_8}{16} e_n n(\log_2(n) - 1),$$

where $\log_2(e_n) = \lceil \log_2(\log_2(n) - 1)\rceil - \log_2(\log_2(n) - 1)$. Notice that $1 \le e_n < 2$.

Finally, to compute the product of degree-n polynomials $f, g \in A[x]$ $(n \ge 4)$, we define $N = 2^{\lfloor \log_2(n)\rfloor+2}$ and compute fg in $A[x]/(x^N + 1)$ at a cost of

$$Cost(n) = \frac{C_8}{4} E_n n(\lfloor \log_2(n)\rfloor + 1),$$

where $\log_2(E_n) = \lfloor \log_2(n)\rfloor - \log_2(n) + \lceil \log_2(\lfloor \log_2(n)\rfloor + 1)\rceil - \log_2(\lfloor \log_2(n)\rfloor + 1)$. Notice that $\frac{1}{2} < E_n < 2$.

In order to illustrate the performance of Schönage-FFT polynomial multiplication, Fig. 4 compares it with the cost of Karatsuba-style method. Anyhow, we did not focus on improving Schönage-FFT method and our experiments are

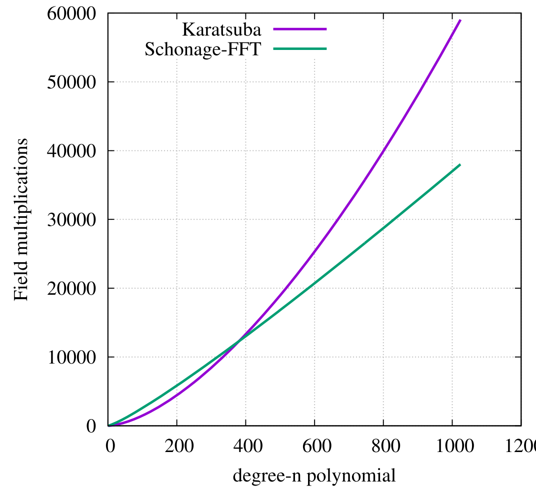


Fig. 4 Comparison between the Schönage-FFT and Karatsuba style polynomial multiplications. The x-axis corresponds with the degree of both polynomials to be multiplied, while y-axis shows the expected cost required in the polynomial multiplication method. In particular, the karatsuba and Schönage-FFT costs are taken as $n^{\log_2(3)}$ and $\frac{27}{8}n(\lfloor \log_2(n)\rfloor + 1)$, respectively. Schönage-FFT method assumes that $E_n = 1/2$, and karatsuba multiplication is required in its base case, which implies $C_8 = 27$

centered on asymtoptic costs. Whichever the case, it looks that Karatsuba-style polynomial multiplication is the more suitable approach to be used in the new $\sqrt{\text{é}}$lu formulas for both as CSIDH and B-SIDH implementations.

A.3 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 $\sqrt{\text{é}}$lu procedures `xISOG` and `xEVAL`.

Formally, each one of the two resultants required by 2 and 3, corresponds to the computation of $\mathrm{Res}_Z(f(Z), g(Z))$ 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 details see [4, p. 27, Sect. 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

$$\begin{aligned}
T(b) &\approx T\left(\frac{b}{2}\right) + 2\left(\frac{b}{2}\right)^{\log_2(3)} + \frac{b}{2} \\
&= T\left(\frac{b}{2}\right) + \frac{2}{3}b^{\log_2(3)} + \frac{b}{2} \\
&= \sum_{i=0}^{\log_2(b)} \left(\frac{2}{3}\left(\frac{b}{2^i}\right)^{\log_2(3)} + \frac{b}{2^{i+1}}\right) \\
&= \left(\frac{2}{3}b^{\log_2(3)}\right)\sum_{i=0}^{\log_2(b)} \frac{1}{3^i} + \left(\frac{b}{2}\right)\sum_{i=0}^{\log_2(b)} \frac{1}{2^i} \\
&= \left(1 - \frac{1}{3^{\log_2(b)+1}}\right) b^{\log_2(3)} + \left(1 - \frac{1}{2^{\log_2(b)+1}}\right) b \\
&= \left(1 - \frac{1}{3b^{\log_2(3)}}\right) b^{\log_2(3)} + \left(1 - \frac{1}{2b}\right) b \\
&= b^{\log_2(3)} + b - \frac{5}{6} .
\end{aligned}$$

Hence, the polynomial reduction $g \bmod f$ is expected to have a running time of $\left(b^{\log_2(3)} + b - \frac{5}{6}\right)$ 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,

$$\begin{aligned}
R(b, i) &= 2^i\left(\left(\frac{b}{2^i}\right)^{\log_2(3)} + \frac{b}{2^i} - \frac{5}{6}\right) \\
&= b^{\log_2(3)}\left(\frac{2}{3}\right)^i + b - \left(\frac{5}{6}\right)2^i .
\end{aligned}$$

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

$$\begin{aligned}
R(b) &= b^{\log_2(3)}\sum_{i=0}^{\log_2(b)}\left(\frac{2}{3}\right)^i + b(\log_2(b)+1) - \frac{5}{6}\sum_{i=0}^{\log_2(b)} 2^i \\
&= 3b^{\log_2(3)}\left(1 - \left(\frac{2}{3}\right)^{\log_2(b)+1}\right) + b(\log_2(b)+1) \\
&\quad - \frac{5}{6}\left(2^{\log_2(b)+1} - 1\right) \\
&= 3b^{\log_2(3)}\left(1 - \frac{2b}{3b^{\log_2(3)}}\right) + b(\log_2(b)+1) \\
&\quad - \frac{5}{6}(2b - 1) \\
&= 3b^{\log_2(3)} - 2b + b\log_2(b) + b - \frac{5}{3}b + \frac{5}{6} \\
&= 3b^{\log_2(3)} + b\log_2(b) - \frac{8}{3}b + \frac{5}{6} .
\end{aligned}$$

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 $\mathtt{Res}_Z(f(Z), g(Z))$ computation requires a total of

$$\left(3b^{\log_2(3)} + b\log_2(b) - \frac{5}{3}b + \frac{5}{6}\right) \text{ field multiplications.}$$

Now, the polynomial $h_I(X)$, which splits into b linear polynomials, is computed via product trees at a cost of

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

multiplications, while $E_{i,J}$ (the product of b quadratic polynomials), requires about

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

A.4 B-SIDH primes

For all primes here we have that $M|(p+1)$ and $N|(p-1)$.

Example 2. of [13, Sect. 5.2] (**B-SIDHp253**):

$$\begin{aligned}
p &= \mathtt{0x1935BECE108DC6C0AAD0712181BB1A414} \\
&\quad\ \mathtt{E6A8AAA6B510FC29826190FE7EDA80F}, \\
M &= 4^2 \cdot 3 \cdot 7^{16} \cdot 17^9 \cdot 31^8 \cdot 311 \cdot 571 \cdot 1321 \cdot 5119 \cdot 6011 \cdot \\
&\quad\ 14207 \cdot 28477 \cdot 76667, \\
N &= 11^{18} \cdot 19 \cdot 23^{13} \cdot 47 \cdot 79 \cdot 83 \cdot 89 \cdot 151 \cdot 3347 \cdot 17449 \cdot
\end{aligned}$$

$$33461 \cdot 51193.$$

Example 3. of [13, Sect. 5.2] (**B-SIDHp255**):

$$\begin{aligned} p &= \texttt{0x76042798BBFB78AEBD02490BD2635DEC} \\ &\quad \texttt{131ABFFFFFFFFFFFFFFFFFFFFFFFFFFF} \\ M &= 4^{55} \cdot 5 \cdot 7^2 \cdot 67 \cdot 223 \cdot 4229 \cdot 9787 \cdot 13399 \cdot 21521 \cdot \\ &\quad 32257 \cdot 47353, \\ N &= 3^{34} \cdot 11 \cdot 17 \cdot 19^2 \cdot 29 \cdot 37 \cdot 53^2 \cdot 97 \cdot 107 \cdot 109 \cdot 131 \cdot \\ &\quad 137 \cdot 197 \cdot 199 \cdot 227 \cdot 251 \cdot 5519 \cdot 9091 \cdot 33997 \cdot \\ &\quad 38201. \end{aligned}$$

Example 5. of [13, Sect. 5.3] (**B-SIDHp247**):

$$\begin{aligned} p &= \texttt{0x46B27D6FAE96ED4A639E045B7D2C3CA33} \\ &\quad \texttt{F476892ADAFF87B9B6EAE5EE1FFFF} \\ M &= \left(4^2 \cdot 5^2 \cdot 7 \cdot 23 \cdot 79 \cdot 107 \cdot 307 \cdot 2129\right)^4 \cdot 7901^2, \\ N &= 3 \cdot 11 \cdot 17 \cdot 241 \cdot 349 \cdot 421 \cdot 613 \cdot 983 \cdot 1327 \cdot 1667 \cdot \\ &\quad 2969 \cdot 3769 \cdot 4481 \cdot 4649 \cdot 4801 \cdot 4877 \cdot 5527 \cdot \\ &\quad 6673 \cdot 7103 \cdot 7537 \cdot 7621. \end{aligned}$$

Example 6. of [13, Sect. 5.3] (**B-SIDHp237**):

$$\begin{aligned} p &= \texttt{0x1B40F93CE52A207249237A4FF37425A798} \\ &\quad \texttt{E914A74949FA343E8EA487FFFF} \\ M &= 4^3 \cdot \left(4 \cdot 3^4 \cdot 17 \cdot 19 \cdot 31 \cdot 37 \cdot 53^2\right)^6, \\ N &= 7 \cdot 13 \cdot 43 \cdot 73 \cdot 103 \cdot 269 \cdot 439 \cdot 881 \cdot 883 \cdot 1321 \cdot \\ &\quad 5479 \cdot 9181 \cdot 12541 \cdot 15803 \cdot 20161 \cdot 24043 \cdot 34843 \cdot \\ &\quad 48437 \cdot 62753 \cdot 72577. \end{aligned}$$

Lucky proposal of [5, appendix A] (**B-SIDHp257**):

$$\begin{aligned} p &= \texttt{0x1E409D8D53CF3BEB65B5F41FB53B25E} \\ &\quad \texttt{BEAF37761CD8BA996684150A40FFFFFFFF} \\ M &= 4^{16} \cdot 5^{21} \cdot 7 \cdot 11 \cdot 163 \cdot 1181 \cdot 2389 \cdot 5233 \cdot 8353 \cdot \\ &\quad 10139 \cdot 11939 \cdot 22003 \cdot 25391 \cdot 41843, \end{aligned}$$

and

$$\begin{aligned} N &= 3^{56} \cdot 31 \cdot 43 \cdot 59 \cdot 271 \cdot 311 \cdot 353 \cdot 461 \cdot 593 \cdot 607 \cdot \\ &\quad 647 \cdot 691 \cdot 743 \cdot 769 \cdot 877 \cdot 1549. \end{aligned}$$

References

1. Adj, G., Cervantes-Vázquez, D., Chi-Domínguez, J., Menezes, A., Rodríguez-Henríquez, F.: On the cost of computing isogenies between supersingular elliptic curves. In: Cid, C., Jacobson, M.J., Jr. (eds.) Selected Areas in Cryptography - SAC 2018–25th International Conference. Lecture Notes in Computer Science, vol. 11349, pp. 322–343. Springer, Cham (2018)
2. Azarderakhsh, R., Campagna, M., Costello, C., De Feo, L., Hess, B., Jalali, A., Jao, D., Koziel, B., LaMacchia, B., Longa, P., Naehrig, M., Pereira, G., Renes, J., Soukharev, V., Urbanik, D.: Supersingular isogeny key encapsulation. second round candidate of the NIST's post-quantum cryptography standardization process (2017). Available at: https://sike.org/
3. Banegas, G., Bernstein, D.J., Campos, F., Chou, T., Lange, T., Meyer, M., Smith, B., Sotáková, J.: CTIDH: faster constant-time CSIDH. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(4), 351–387 (2021)
4. Bernstein, D.J.: Fast multiplication and its applications. Algorithmic Number Theory **44**, 325–384 (2008)
5. Bernstein, D.J., De Feo, L., Leroux, A., Smith, B.: Faster computation of isogenies of large prime degree. In: ANTS XIV. The Open Book Series, vol. 4(1), pp. 39–55 (2020)
6. Bernstein, D.J., Lange, T., Martindale, C., Panny, L.: Quantum circuits for the CSIDH: optimizing quantum evaluation of isogenies. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology—EUROCRYPT 2019, Part II. Lecture Notes in Computer Science, vol. 11477, pp. 409–441. Springer, Cham (2019)
7. Bonnetain, X., Schrottenloher, A.: Quantum security analysis of CSIDH. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology—EUROCRYPT 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12106, pp. 493–522. Springer, Cham (2020)
8. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: an efficient post-quantum commutative group action. In: Peyrin, T., Galbraith, S.D. (eds.) Advances in Cryptology—ASIACRYPT 2018, Part III. Lecture Notes in Computer Science, vol. 11274, pp. 395–427. Springer, Cham (2018)
9. Cervantes-Vázquez, D., Chenu, M., Chi-Domínguez, J., De Feo, L., Rodríguez-Henríquez, F., Smith, B.: Stronger and faster side-channel protections for CSIDH. In: Schwabe, P., Thériault, N. (eds.) Progress in Cryptology—LATINCRYPT 2019. Lecture Notes in Computer Science, vol. 11774, pp. 173–193. Springer, Cham (2019)
10. Chávez-Saab, J., Chi-Domínguez, J., Jaques, S., Rodríguez-Henríquez, F.: The SQALE of CSIDH: sublinear Vélu quantum-resistant isogeny action with low exponents. J. Cryptogr. Eng. (2021). https://link.springer.com/article/10.1007/s13389-021-00271-w
11. Chi-Domínguez, J., Rodríguez-Henríquez, F.: Optimal strategies for CSIDH. Advances in Mathematics of Communications (2020). https://www.aimsciences.org/article/doi/10.3934/amc.2020116. Preprint version: https://eprint.iacr.org/2020/417
12. Connolly, D.: Code for SIDH key exchange with optional public key compression. Github (2017). Available at: https://github.com/dconnolly/msr-sidh/tree/master/SIDH-Magma
13. Costello, C.: B-SIDH: supersingular isogeny Diffie-Hellman using twisted torsion. In: Moriai, S., Wang, H. (eds.) Advances in Cryptology—ASIACRYPT 2020—Proceedings, Part II. Lecture Notes in Computer Science, vol. 12492, pp. 440–463. Springer, Cham (2020)
14. Costello, C., Hisil, H.: A simple and compact algorithm for SIDH with arbitrary degree isogenies. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology—ASIACRYPT 2017, Part II. Lecture Notes in Computer Science, vol. 10625, pp. 303–329. Springer, Cham (2017)
15. Costello, C., Longa, P., Naehrig, M.: Efficient algorithms for supersingular isogeny Diffie-Hellman. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology—CRYPTO 2016, pp. 572–601. Springer, Berlin Heidelberg, Berlin, Heidelberg (2016)
16. Couveignes, J.M.: Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291 (2006). http://eprint.iacr.org/2006/291

17. De Feo, L., Jao, D., Plût, J.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. J. Math. Cryptol. **8**(3), 209–247 (2014)
18. De Feo, L., Kieffer, J., Smith, B.: Towards practical key exchange from ordinary isogeny graphs. In: Peyrin, T., Galbraith, S.D. (eds.) Advances in Cryptology—ASIACRYPT 2018, Part III. Lecture Notes in Computer Science, vol. 11274, pp. 365–394. Springer, Cham (2018)
19. Galbraith, S.D., Petit, C., Shani, B., Ti, Y.B.: On the security of supersingular isogeny cryptosystems. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology—ASIACRYPT 2016, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10031, pp. 63–91. Springer, Berlin (2016)
20. Hutchinson, A., LeGrow, J.T., Koziel, B., Azarderakhsh, R.: Further optimizations of CSIDH: a systematic approach to efficient strategies, permutations, and bound vectors. In: Conti, M., Zhou, J., Casalicchio, E., Spognardi, A. (eds.) Applied Cryptography and Network Security—18th International Conference, ACNS 2020, Part I. Lecture Notes in Computer Science, vol. 12146, pp. 481–501. Springer, Cham (2020)
21. Jao, D., De Feo, L.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: Yang, B. (ed.) Post-Quantum Cryptography–4th International Workshop, PQCrypto 2011. Lecture Notes in Computer Science, vol. 7071, pp. 19–34. Springer, Berlin (2011)
22. Kohel, D.R.: Endomorphism rings of elliptic curves over finite fields. Ph.D. thesis, University of California at Berkeley, The address of the publisher (1996). Available at:http://iml.univ-mrs.fr/~kohel/pub/thesis.pdf
23. Meyer, M.: Isogeny School 2020: Constant-time implementations of isogeny schemes. Isogeny-based cryptography school, Week 11 (2020). https://isogenyschool2020.co.uk/schedule/isogenyschool-constant-time.pdf
24. Meyer, M., Campos, F., Reith, S.: On lions and elligatorsaefficient constant-time implementation of CSIDH. In: Ding, J., Steinwandt, R. (eds.) Post-Quantum Cryptography—0th International Conference. Lecture Notes in Computer Science, vol. 11505, pp. 307–325. Springer, Cham (2019)
25. Meyer, M., Reith, S.: A faster way to the CSIDH. In: INDOCRYPT 2018, Lecture Notes in Computer Science, vol. 11356, pp. 137–152. Springer (2018)
26. Montgomery, P.L.: Speeding the Pollard and elliptic curve methods of factorization. Math. Comput. **48**(177), 243–264 (1987)
27. Moody, D., Shumow, D.: Analogues of Vélu's formulas for isogenies on alternate models of elliptic curves. Math. Comput. **85**(300), 1929–1951 (2016)
28. Nakagawa, K., Onuki, H., Takayasu, A., Takagi, T.: L_1-norm ball for CSIDH: optimal strategy for choosing the secret key space. IACR Cryptol. ePrint Arch. p. 181 (2020)
29. Onuki, H., Aikawa, Y., Yamazaki, T., Takagi, T.: (short paper) A faster constant-time algorithm of CSIDH keeping two points. In: Attrapadung, N., Yagi, T. (eds.) 14th International Workshop on Security, IWSEC 2019. Lecture Notes in Computer Science, vol. 11689, pp. 23–33. Springer, Cham (2019)
30. Peikert, C.: He gives c-sieves on the CSIDH. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology—EUROCRYPT 2020—Proceedings, Part II. Lecture Notes in Computer Science, vol. 12106, pp. 463–492. Springer, Berlin (2020)
31. Renes, J.: Computing isogenies between Montgomery curves using the action of (0, 0). In: Lange, T., Steinwandt, R. (eds.) Post-Quantum Cryptography—9th International Conference, PQCrypto 2018. Lecture Notes in Computer Science, vol. 10786, pp. 229–247. Springer, Cham (2018)
32. Rostovtsev, A., Stolbunov, A.: Public-key cryptosystem based on isogenies. IACR Cryptology ePrint Archive, vol. 2006, p. 145 (2006). http://eprint.iacr.org/2006/145
33. Schönhage, A.: Schnelle multiplikation von polynomen über körpern der charakteristik 2. Acta Informatica **7**, 395–398 (1977)
34. Stolbunov, A.: Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves. Adv. Math. Commun. **4**(2), 215–235 (2010)
35. Washington, L.: Elliptic Curves: Number Theory and Cryptography, 2nd edn. Chapman & Hall/CRC, Boca Raton (2008)