

# DeCSIDH: Delegating isogeny computations in the CSIDH setting

Robi Pedersen<sup>[0000–0001–5120–5709]</sup>

imec-COSIC KU Leuven  
Kasteelpark Arenberg 10 Bus 2452, 3001 Leuven, Belgium  
`robi.pedersen@esat.kuleuven.be`

**Abstract.** Delegating heavy computations to auxiliary servers, while keeping the inputs secret, presents a practical solution for computationally limited devices to use resource-intense cryptographic protocols, such as those based on isogenies, and thus allows the deployment of post-quantum security on mobile devices and in the internet of things. We propose two algorithms for the secure and verifiable delegation of isogeny computations in the CSIDH setting. We then apply these algorithms to different instances of CSIDH and to the signing algorithms SeaSign and CSI-FiSh. Our algorithms present a communication-cost trade-off. Asymptotically (for high communication), the cost for the delegator is reduced by a factor 9 for the original CSIDH-512 parameter set and a factor 30 for SQALE'd CSIDH-4096, while the relative cost of SeaSign vanishes. Even for much lower communication cost, we come close to these asymptotic results. Using the knowledge of the class group, the delegation of CSI-FiSh is basically free (up to element generation) already at a very low communication cost.

**Keywords:** Post-quantum cryptography · Isogeny-based cryptography · CSIDH · Secure computation outsourcing · Lightweight cryptography

## 1 Introduction

*Delegation of Computations.* The last decade has witnessed an immense surge in mobile devices, including RFID-cards, tiny sensor nodes, smart phones and a myriad of devices in the internet of things. Since such mobile devices are usually computationally limited or have other constraints such as low battery life, the delegation of their computations to external, more powerful devices, has become an active area of research. While delegation allows to relieve these devices of their most heavy computations, it comes at a certain risk, such as potentially malicious servers trying to extract sensitive data or returning wrong results for these computations. Mitigating these threats is especially important when delegating cryptographic protocols, where such servers might try to extract private keys. The necessary properties for secure and verifiable delegation were first formalized in a security model introduced by Hohenberger and Lysyanskaya [15]

in the context of group exponentiations. Their model lets the delegator shroud sensitive data before sending it to the server and then verify and de-shroud the server’s output. The operations performed by the delegator should still be efficient enough for the delegation to be worthwhile.

*Isogeny-based Cryptography.* Isogeny-based cryptography goes back to the works of Couveignes [12] and Rostovtsev and Stolbunov [24] and is based on the difficulty of finding an explicit isogeny linking two given isogenous elliptic curves defined over a finite field. While the original proposal uses ordinary elliptic curves, recent quantum attacks [11, 18, 23], which use the commutativity of the endomorphism ring, push the secure parameter size to the realm of prohibitively inefficient protocols. In response, two new approaches using supersingular elliptic curves have been introduced. The first one, commonly referred to as SIDH (supersingular isogeny Diffie-Hellman) was proposed by Jao and De Feo [16] and uses the fact that supersingular elliptic curves over  $\mathbb{F}_{p^2}$  have a non-commutative endomorphism ring, so that the previously discussed attacks are not applicable. The second one, called CSIDH [7] (commutative SIDH), uses the structure of supersingular elliptic curves to immensely reduce the computational cost of the originally proposed protocols back to the realm of usability. We note that while CSIDH closely follows the line of the original Couveignes-Rostovtsev-Stolbunov scheme, SIDH uses a different approach that is more closely related to the cryptographic hash function proposed by Charles, Goren and Lauter [8].

*Motivation and Related Work.* While isogeny-based protocols profit from the lowest key sizes of any of the current post-quantum standardization proposals [1, 7, 16, 26, 19], they are still among the slowest. This might be tolerable for specific applications, but given the immense surge in low-power mobile devices in recent years, there is a strong need for easily deployable and computationally cheap, yet secure cryptographic protocols. It is of particular interest for these limited devices to profit from post-quantum security in order to allow them to remain secure in the long term. While there have been many proposals for the delegation of group exponentiations and pairings [15, 29], the delegation of post-quantum cryptographic protocols is a very new topic. In 2019, Pedersen and Uzunkol [20] proposed the first delegation algorithms for isogeny computations and improved upon their work with a follow-up paper in 2021 [21]. Their approach is applied to SIDH-type protocols, i.e. supersingular isogeny protocols over  $\mathbb{F}_{p^2}$ , and is based on the outsource security model from [15]. The question of delegating isogenies in the CSIDH setting has been proposed as a direction of future research by [21] and will be the main focus of this work. While we will also use the outsource security model from [15], we stress that we cannot simply use or translate the previously proposed isogeny delegation schemes in the SIDH setting to the CSIDH setting. The main reason is that cryptographic protocols in these two schemes use very different descriptions and are not related to one another in an obvious way.

*Our Contribution.* The purpose of this work is to propose the first isogeny delegation algorithms in the CSIDH setting, which are secure and provide high verifiability guarantees. More precisely,

1. We introduce and analyze **ShrVec**, an algorithm that allows transforming a uniform vector into three vectors, two of which are uniform, and the third one being small. This allows to shroud secret keys in the CSIDH protocol [7].
2. We define two new algorithms based on the outsource-security description of Hohenberger and Lysyanskaya [15]:
  - The (commutative) *isogeny computation algorithm* **Clso**, which allows to delegate the computation of an isogeny, while keeping the kernel hidden from the auxiliary servers, and
  - The *hidden isogeny computation algorithm* **Hlso**, which allows to delegate the computation of an isogeny, while keeping both the kernel and the isogeny codomain hidden from the auxiliary servers.

We present both algorithms in the *one-malicious two untrusted program* (OMTUP) assumption defined in [15] and in the newly introduced *two honest-but-curious* (2HBC) assumption. All of our algorithms work in two rounds of communication.

3. We apply our delegation algorithms to different protocols in the CSIDH setting and compare the reduced cost of the delegator to the full, local computation. Most of our algorithms allow a trade-off between computational and communication cost. Asymptotically (for large communication cost), we reduce the computational cost of CSIDH-512 [7] to below 12% of the local cost of the full protocol, while the SQALE'd CSIDH-4096 [9] protocol can be reduced to about 3.5% of the local cost. Also for lower communication costs, the gain of the delegator quickly approaches the asymptotic values. The gains for signatures are even better: The relative cost of delegating SeaSign asymptotically vanishes and can be easily reduced to a few percent at low communication cost, while CSI-FiSh, by using knowledge of the class group structure, can be made virtually free at low communication cost.

*Naming.* Following the *fishy name trend* of commutative supersingular isogeny protocols, we refer to their delegation as DeCSIDH (**D**elegated **C**SIDH) and pronounce it *decks*ide. The reader is free to imagine a fisher with limited resources being helped by a more powerful (yet potentially malicious) fishing boat.

**Acknowledgments.** This work was supported in part by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101020788 - Adv-ERC-ISOCRYPT), the Research Council KU Leuven grant C14/18/067, and by CyberSecurity Research Flanders with reference number VR20192203. The author would like to thank Frederik Vercauteren and Osmanbey Uzunkol, as well as the anonymous reviewers for valuable feedback regarding this work.

## 2 Elliptic Curves and Isogenies

Isogeny-based cryptography is based on the good mixing properties of isogeny graphs, i.e. graphs of isomorphism classes of elliptic curves over finite fields

connected by isogenies. Isogenies are surjective homomorphisms between elliptic curves that are also algebraic maps. Separable isogenies are uniquely defined by their kernel. While it is easy to compute an isogeny from a given kernel, it is in general difficult to find the kernel, given two isogenous elliptic curves.

The original protocols by Couveignes [12] and Rostovtsev and Stolbunov [24, 27] used ordinary elliptic curves, defined over a prime field  $\mathbb{F}_p$ , while the later CSIDH protocol by Castryck, Lange, Martindale, Panny and Renes [6] uses supersingular elliptic curves over  $\mathbb{F}_p$  for efficiency reasons. These curves have Frobenius trace  $t = 0$  and their  $\mathbb{F}_p$ -rational endomorphism rings are orders  $\mathcal{O}$  in a quadratic imaginary field  $\mathbb{Q}(\sqrt{-p})$ . A key observation of these protocols is that the ideals in the class group  $\text{Cl}(\mathcal{O})$  uniquely define subgroups via their kernel and therefore uniquely define isogenies, i.e. for a given elliptic curve  $E/\mathbb{F}_p$  and ideal  $\mathfrak{a} \in \text{Cl}(\mathcal{O})$ , we have a separable isogeny  $E \rightarrow E/\mathfrak{a}$  with kernel  $\bigcap_{\alpha \in \mathfrak{a}} \ker \alpha$ . As a result, the ideal-class group  $\text{Cl}(\mathcal{O})$  acts freely and transitively on the set of  $\mathbb{F}_p$ -isomorphism classes of these elliptic curves via isogenies [12] and this group action is generally written as  $E \rightarrow \mathfrak{a} * E$ .

In the CSIDH protocol [7], the underlying prime field  $\mathbb{F}_p$  is defined via  $p = 4 \prod_{i=1}^n \ell_i - 1$ , where the  $\ell_i$  are small primes. Since  $\#E(\mathbb{F}_p) = p + 1$ , the chosen structure of  $p$  implies that  $\ell_i \mathcal{O}$  decomposes as the product of two prime ideals  $\mathfrak{l}_i = (\ell_i, \pi - 1)$  and  $\mathfrak{l}_i^{-1} = (\ell_i, \pi + 1)$ , where  $\pi$  corresponds to the Frobenius endomorphism. The action of these ideals on the set of (isomorphism classes of) elliptic curves over  $\mathbb{F}_p$  can then be computed with the standard Vélú formulae [30] and are efficient for small  $\ell_i$ . Given the structure of  $p$ , ideals can generally be expressed as  $\mathfrak{a} = \prod_{i=1}^n \mathfrak{l}_i^{a_i}$ , where positive exponents  $a_i$  correspond to the action of  $\mathfrak{l}_i$ , while negative exponents correspond to the action of  $\mathfrak{l}_i^{-1}$ . Ideals can then be simply expressed by representative vectors, e.g.  $\mathfrak{a} = (a_1, \dots, a_n)$  would correspond to the action of  $\mathfrak{a}$  as defined above. The order of the application of the prime ideals  $\mathfrak{l}_i$  of  $\mathfrak{a}$  does not matter and its dual is simply  $\mathfrak{a}^{-1}$  represented by  $-\mathfrak{a}$ . Note that  $\mathfrak{a}_1 \mathfrak{a}_2$  corresponds to  $\mathfrak{a}_1 + \mathfrak{a}_2$ .

Isogenies can be computed using Algorithm 2 of [7]. We denote by  $\mathcal{I}(\mathfrak{a})$  the generic cost of computing an isogeny defined by the ideal  $\mathfrak{a}$ .

*The Class Group.* While the class group has asymptotic size  $\#\text{Cl}(\mathcal{O}) \approx 2\sqrt{p}$  [25], computing its exact structure is a difficult task for large  $p$  [3, 17]. The original proposal of CSIDH-512 [7] circumvented this problem by choosing  $n = 74$  small primes (the 73 smallest odd primes and  $\ell_{74} = 587$ ) and sampling the elements  $a_i$  of  $\mathfrak{a}$  from a range  $\{-5, \dots, 5\}$  of size 11. As such,  $11^n \approx 2^{256}$ , which should cover most of the class group without knowing its exact structure. In 2019, Beullens, Kleinjung and Vercauteren [3] computed the class group structure and the relation lattice for the CSIDH-512 parameter set and found a cyclic class group of order  $\#\text{Cl}(\mathcal{O}) \approx 2^{257}$ . This knowledge allows to sample random elements from  $\mathbb{Z}/\#\text{Cl}(\mathcal{O})\mathbb{Z} = \mathbb{Z}/\#\text{Cl}(\mathcal{O})\mathbb{Z}$  and transform them into vectors  $\mathfrak{a}$  by solving easy instances of the closest vector problem using the relation lattice. This guarantees uniform coverage of the entire class group, while also allowing efficient computation via low-degree isogenies. Unfortunately, class group computations for larger parameter sets than CSIDH-512 seem currently out of reach.

*Notation.* We use “ $\leftarrow$ ” as the assignment operator: If the right hand side is an algorithm, the left hand side represents the variables to which its output is assigned. If the right hand side is a set, we assume the left hand side to represent a randomly sampled value from this set. We will write  $[start, end]$  as a shorthand for the set of integers ranging from  $start \in \mathbb{Z}$  to  $end \in \mathbb{Z}$ . We define as  $\mathbb{B}(N) \subset \mathbb{Z}^n$  any set of the form  $\mathbb{B} = \mathbb{B}_1 \times \cdots \times \mathbb{B}_n$ , where  $\mathbb{B}_i \subset \mathbb{Z}$  are intervals of length  $d_i = \#\mathbb{B}_i$ , and such that  $\#\mathbb{B}(N) = \prod_{i=1}^n d_i \approx N$ . As an example, for CSIDH-512, we use  $\mathbb{B}(2^{256}) = [-5, 5]^n$ . Ideals in  $\text{Cl}(\mathcal{O})$  can then be represented by vectors  $\mathbf{a} \in \mathbb{B}(N)$ , where typically  $N \leq \#\text{Cl}(\mathcal{O})$ . Intervals  $\mathbb{B}_i$  are of the types  $[-B_i, B_i]$  or  $[0, B_i]$  for  $B_i \in \mathbb{N}$  (see e.g. [5, 7, 9]). Throughout this work, we will use the former case for simplicity, for which it holds  $d_i = 2B_i + 1$ . The case  $[0, B_i]$  follows completely analogously.

We write ideals in  $\text{Cl}(\mathcal{O})$  in the fraktur font (e.g.  $\mathfrak{a}, \mathfrak{b}, \mathfrak{s}, \dots$ ) while the corresponding vectors in  $\mathbb{B}(N)$  are written in bold font (e.g.  $\mathbf{a}, \mathbf{b}, \mathbf{s}, \dots$ ). If the class group is known, we write elements from  $\mathbb{Z}_{\#\text{Cl}(\mathcal{O})}$  in the standard font (e.g.  $a, b, s, \dots$ ). We assume  $\text{Cl}(\mathcal{O})$  to be cyclic with publicly known generator  $\mathfrak{g}$ .<sup>1</sup> We always see elements using the same letters as related, e.g.  $a \in \mathbb{Z}_{\#\text{Cl}(\mathcal{O})}$  and  $\mathbf{a} \in \mathbb{B}(N)$  will always represent  $\mathfrak{a} \in \text{Cl}(\mathcal{O})$ , while the same holds for  $(b, \mathbf{b}, \mathfrak{b})$ ,  $(s, \mathbf{s}, \mathfrak{s})$  etc. Let  $\mathbf{a} = (a_1, \dots, a_n)$ , then we can express this relation as follows:

$$\mathbf{a} = \prod_{i=1}^n t_i^{a_i} = \mathfrak{g}^{\mathbf{a}}.$$

Note that vector entries are also written in the standard font. Their distinction from elements in  $\mathbb{Z}_{\#\text{Cl}(\mathcal{O})}$  will always be clear from context.

It is useful to note that multiplications between elements in  $\text{Cl}(\mathcal{O})$  naturally translate to additions in  $\mathbb{Z}_{\#\text{Cl}(\mathcal{O})}$  and  $\mathbb{B}(N)$ , while divisions translate to subtractions. As an example,  $\mathfrak{a}\mathfrak{b}^{-1}$  can be represented by  $a - b$  or by  $\mathbf{a} - \mathbf{b}$ .

*Security.* Security of CSIDH and related protocols is generally based on the following hard problem.

**Definition 1 (Group action inverse problem (GAIP)).** [7] *Given two supersingular elliptic curves  $E, E'$  over  $\mathbb{F}_p$  with the same  $\mathbb{F}_p$ -rational endomorphism ring  $\mathcal{O}$ , find an ideal  $\mathfrak{a} \in \text{Cl}(\mathcal{O})$  such that  $E' = \mathfrak{a} * E$ .*

Classical security is based on a meet-in-the-middle attack. The query complexity of this attack is  $O(\sqrt{\#\text{Cl}(\mathcal{O})})$ . Quantum security of CSIDH is still subject to scrutiny. For current estimates of the quantum security, we refer the reader to [4, 7, 9] and [22]. We will use these estimates for later assessment of our schemes and always refer to the source in question. We write  $\lambda(N)$  for a generic quantum security parameter for a class group of size approximately  $N$ .

<sup>1</sup> Throughout this work, we will only consider the known class group established in [3]. In any other case, where  $\text{Cl}(\mathcal{O})$  would not be cyclic, we can always assume to work in a cyclic subgroup. For simplicity, we will still refer to it as the class group and write  $\text{Cl}(\mathcal{O})$ .

### 3 Secure and Verifiable Delegation

#### 3.1 Security Model by Hohenberger and Lysyanskaya

The secure delegation model of Hohenberger and Lysyanskaya [15] is defined around three central entities: a delegator  $\mathcal{T}$ , a set of auxiliary servers  $\mathcal{U}$  and the environment  $\mathcal{E}$ . The delegator interacts with the servers, denoted as  $\mathcal{T}^{\mathcal{U}}$ , so that they jointly implement an algorithm Alg at a lower computational cost for  $\mathcal{T}$ , than if  $\mathcal{T}$  would run Alg itself. The environment represents any third party, that might observe the interaction or that might later (or previously) interact with  $\mathcal{T}$  itself. Most notably,  $\mathcal{E}$  includes the manufacturer of the service provided by  $\mathcal{U}$ . A key assumption of the model is that after  $\mathcal{T}$  starts using  $\mathcal{U}$ , there is no more direct channel between  $\mathcal{U}$  and  $\mathcal{E}$  or between the different servers in  $\mathcal{U}$ . The rationale behind this, is that  $\mathcal{T}$  has access to  $\mathcal{U}$  only through a firewall. Yet, these entities can still try to communicate indirectly. Thus, this interaction has multiple threats to mitigate: First,  $\mathcal{T}$  has to make sure that neither  $\mathcal{E}$  nor  $\mathcal{U}$  gain any sensitive information from  $\mathcal{T}$ 's interaction with  $\mathcal{U}$  (and possibly later with  $\mathcal{E}$ ). In general, this means that  $\mathcal{T}$  has to find a way to shroud sensitive data before passing it on to  $\mathcal{U}$  and be able to recover its desired result (i.e. the output of Alg) from whatever  $\mathcal{U}$  returns. Secondly, to be able to do so,  $\mathcal{T}$  also needs a way to verify that the output of  $\mathcal{U}$  is indeed correct. This is generally achieved by checking that the outputs fulfill some verification conditions that adversarially produced outputs could only fulfill with a low probability.

The following definition summarizes the security assumptions used throughout this work and includes the reduction in computational cost  $\alpha$  that  $\mathcal{T}$  profits from, when compared to the local computation, as well as the degree of certainty  $\beta$  that the outputs of the servers are correct.

**Definition 2** ( $(\alpha, \beta)$ -outsourcing-security). [15] *Let Alg be an algorithm with the following outsource input/output specification: We distinguish secret, protected and unprotected inputs and outputs, depending on whether only  $\mathcal{T}$  has access, only  $\mathcal{T}$  and  $\mathcal{E}$  have access, or all parties have access, respectively. The non-secret inputs are further subdivided into honest and adversarial, depending on whether they originate from a trusted source or not. Then, the pair  $(\mathcal{T}, \mathcal{U})$  constitutes an  $(\alpha, \beta)$ -outsourcing-secure implementation of Alg if:*

- **Correctness:**  $\mathcal{T}^{\mathcal{U}}$  is a correct implementation of Alg.
- **Security:** For all PPT adversaries  $\mathcal{A} = (\mathcal{E}, \mathcal{U})$ , there exist PPT simulators  $(\mathcal{S}_1, \mathcal{S}_2)$  that can simulate the views of  $\mathcal{E}$  and  $\mathcal{U}$  indistinguishable from the real process. If  $\mathcal{U}$  consists of multiple servers  $\mathcal{U}_i$ , then there is a PPT-simulator  $\mathcal{S}_{2,i}$  for each of their views. We formalize this with the following pairs:
  - **Pair One:**  $\mathcal{E}VIEW_{real} \sim \mathcal{E}VIEW_{ideal}$ :  $\mathcal{E}$  learns nothing about the secret inputs and outputs.
  - **Pair Two:**  $\mathcal{U}VIEW_{real} \sim \mathcal{U}VIEW_{ideal}$ :  $\mathcal{U}$  learns nothing about the secret and (honest/adversarial) protected inputs and outputs.

For a more formal description of these experiments, we refer the reader to Definition 2.2 of [15].

- for all inputs  $x$ , the running time of  $\mathcal{T}$  is at most an  $\alpha$ -multiplicative factor of the running time of  $\text{Alg}(x)$  (i.e.  $\text{Time}(\mathcal{T}) \leq \alpha \text{Time}(\text{Alg})$ ),
- for all inputs  $x$ , if  $\mathcal{U}$  deviates from its advertised functionality during the execution of  $\mathcal{T}^{\mathcal{U}}(x)$ , then  $\mathcal{T}$  will detect the error with probability  $\geq \beta$ .

We call  $\alpha$  the *cost reduction function* and  $\beta$  the *verifiability* of a delegation algorithm. Many adversarial models for  $\mathcal{U}$  have been proposed in the literature, differing along the number of servers and their adversarial powers. In this work, we will use the OMTUP and 2HBC assumptions, the latter being based on the one-server honest-but-curious assumption from [10].

**Definition 3 (OMTUP [15]).** *The one-malicious version of a two untrusted program model defines the adversary as  $\mathcal{A} = (\mathcal{E}, (\mathcal{U}_1, \mathcal{U}_2))$  and assumes that at most one of the two servers  $\mathcal{U}_1$  or  $\mathcal{U}_2$  deviates from its advertised functionality (for a non-negligible fraction of the inputs), while  $\mathcal{T}$  does not know which one.*

**Definition 4 (2HBC).** *The two honest-but-curious program model defines the adversary as  $\mathcal{A} = (\mathcal{E}, (\mathcal{U}_1, \mathcal{U}_2))$ , where  $\mathcal{U}_1$  and  $\mathcal{U}_2$  are servers that always return correct results, but may try to extract sensitive data.*

### 3.2 Advertised Server Functionality

For our purposes throughout this work, we assume that as input, we give the servers multiple pairs  $(\mathbf{a}_1, E_1), \dots, (\mathbf{a}_k, E_k)$  consisting of ideals  $\mathbf{a}_i$  and associated elliptic curves  $E_i$ . The servers then generate and return the codomain curves  $\mathbf{a}_i * E_i$  for each  $i = 1, \dots, k$ . We write

$$(\mathbf{a}_1 * E_1, \dots, \mathbf{a}_k * E_k) \leftarrow \mathcal{U}((\mathbf{a}_1, E_1), \dots, (\mathbf{a}_k, E_k)).$$

We assume that the input elements are always given in a random order as to avoid distinguishability of the elements. We define two ways for the delegator to transmit ideals to the server:

- In the case where  $\text{Cl}(\mathcal{O})$  is known with generator  $\mathbf{g}$ , we assume that we can give an element  $a \in \mathbb{Z}_{\# \text{Cl}(\mathcal{O})}$  to the server, which represents the ideal  $\mathbf{a} = \mathbf{g}^a$ . The server can efficiently compute a short representation of  $\mathbf{a}$  using the relation lattice by applying the procedure described in [3].
- Otherwise, the delegator can give a vector  $\mathbf{a} \in \mathbb{B}$ , representing  $\mathbf{a} = \prod_{i=1}^n \mathfrak{l}_i^{a_i}$ , to the servers.

## 4 Shrouding and Splitting

Before we present implementations for our delegation algorithms, we discuss how to shroud ideals. The basic idea is to split the secret  $\mathbf{s}$  into a pair of random-looking ideals  $(\mathbf{a}_1, \mathbf{a}_2)$ , so that  $\mathbf{a}_1 * (\mathbf{a}_2 * E) = \mathbf{s} * E$ . In the case where  $\text{Cl}(\mathcal{O})$  is known, we can simply generate  $(a, s - a)$  for  $a \leftarrow \mathbb{Z}_{\# \text{Cl}(\mathcal{O})}$ . If  $\text{Cl}(\mathcal{O})$  is unknown, on the other hand, we cannot simply generate  $(\mathbf{a}, \mathbf{s} - \mathbf{a})$  for a random vector

$\mathbf{a} = (a_1, \dots, a_n) \in \mathbb{B}$  since  $\mathbf{s} - \mathbf{a}$  would no longer be in  $\mathbb{B}$  and leak information about the secret [28]. A similar problem was addressed in [13] using rejection sampling: taking vector elements  $a_i \leftarrow [-(\delta_i + 1)B_i, (\delta_i + 1)B_i]$  for integers  $\delta_i \geq 1$ , so that  $s_i - a_i \in [-\delta_i B_i, \delta_i B_i]$  for all  $i \in \{1, \dots, n\}$  makes  $\mathbf{s} - \mathbf{a}$  look uniform. On the other hand,  $\mathbf{a}$  is then no longer uniformly distributed in  $\mathbb{B}(N)$ , since e.g.  $s_i = -B$  would exclude the values of  $a_i > (\delta - 1)B$ . This is not an issue in [13], since  $\mathbf{a}$  is never directly revealed. In our case, however, we also want to delegate the computation of the isogeny defined by  $\mathbf{a}$ , and currently this would reveal information about the secret. We circumvent this problem in Algorithm 1 by splitting up  $\mathbf{s}$  into three vectors  $\mathbf{r}_0, \mathbf{r}_1, \mathbf{r}^*$ , so that the first two are uniform, while the third one contains extra information about  $\mathbf{s}$  that the delegator computes itself. To make  $\mathbf{r}_0$  and  $\mathbf{r}_1$  uniform,  $\mathbf{r}^*$  is in general non-zero. The goal of the algorithm is to minimize the Hamming weight of  $\mathbf{r}^*$ . We define by  $\chi(k)$  the uniform distribution in  $[-k, k]$  and let  $\delta\mathbb{B}(N) = [-\delta_1 B_1, \delta_1 B_1] \times \dots \times [-\delta_n B_n, \delta_n B_n]$ . We further present Algorithm 2, which allows to split a vector  $\mathbf{s}$  into two vectors  $\mathbf{s}'$  and  $\mathbf{s}^*$ , so that  $\mathbf{s}^*$  has a given Hamming weight.

<p><b>Input</b> : secret <math>\mathbf{s} = (s_1, \dots, s_n) \in \mathbb{B}</math> and parameters <math>\delta = (\delta_1, \dots, \delta_n)</math>  <b>Output</b>: <math>\mathbf{r}_0, \mathbf{r}_1 \in \delta\mathbb{B}</math> uniform, <math>\mathbf{r}^* \in \mathbb{B}</math> small, such that <math>\mathbf{r}_0 + \mathbf{r}_1 + \mathbf{r}^* = \mathbf{s}</math></p> <pre style="margin: 0;"> 1 <b>for</b> <math>i = 1, \dots, n</math> <b>do</b> 2   <b>repeat</b> 3     <math>r_{0,i} \leftarrow \chi((\delta_i + 1)B_i)</math> 4     <math>r_{1,i} = s_i - r_{0,i}</math> 5   <b>until</b> <math> r_{0,i}  \leq \delta_i B_i</math> <b>or</b> <math> r_{1,i}  \leq \delta_i B_i</math> 6   <math>b \leftarrow \{0, 1\}</math> 7   <b>if</b> <math> r_{0,i}  &gt; \delta_i B_i</math> <b>then</b> 8     <b>if</b> <math>b == 0</math> <b>then</b> <math>r_{1,i} \leftarrow \chi(\delta_i B_i)</math> 9     <math>r_{0,i} = -r_{1,i}</math> 10    <math>r_i^* = s_i</math> 11  <b>else if</b> <math> r_{1,i}  &gt; \delta_i B_i</math> <b>then</b> 12    <b>if</b> <math>b == 0</math> <b>then</b> <math>r_{0,i} \leftarrow \chi(\delta_i B_i)</math> 13    <math>r_{1,i} = -r_{0,i}</math> 14    <math>r_i^* = s_i</math> 15  <b>else</b> <math>r_i^* = 0</math> 16 <b>end</b> 17 <b>return</b> <math>\mathbf{r}_0 = (r_{0,1}, \dots, r_{0,n}), \mathbf{r}_1 = (r_{1,1}, \dots, r_{1,n}), \mathbf{r}^* = (r_1^*, \dots, r_n^*)</math>.</pre>
---

**Algorithm 1:** ShrVec: Shrouding a vector in  $\mathbb{B}$ .

We write the invocation of these algorithms as  $(\mathbf{r}_0, \mathbf{r}_1, \mathbf{r}^*) \leftarrow \text{ShrVec}_\delta(s)$  and  $(\mathbf{s}^*, \mathbf{s}') \leftarrow \text{Split}(\mathbf{s}, k)$ , respectively. We generally omit  $\delta$  in the index if it is clear from the context or not explicitly needed. In Section A of the supporting material, we prove correctness of our algorithms and the two Lemmas below. We

<p><b>Input</b> : secret <math>\mathbf{s} = (s_1, \dots, s_n) \in \mathbb{B}</math>, and parameter <math>k</math></p> <p><b>Output</b>: <math>\mathbf{s}', \mathbf{s}^* \in \mathbb{B}</math>, such that <math>\mathbf{s}^*</math> has Hamming weight <math>\leq k</math> and <math>\mathbf{s}' + \mathbf{s}^* = \mathbf{s}</math>.</p> <p>1 Sample a uniform subset <math>C^* \leftarrow \{1, \dots, n\}</math> of size <math>k</math>.</p> <p>2 <b>for</b> <math>i = 1, \dots, n</math> <b>do</b></p> <p>3     <b>if</b> <math>i \in C^*</math> <b>then</b> <math>(s_i^*, s_i') = (s_i, 0)</math></p> <p>4     <b>else</b> <math>(s_i^*, s_i') = (0, s_i)</math></p> <p>5 <b>end</b></p> <p>6 <b>return</b> <math>\mathbf{s}^* = (s_1^*, \dots, s_n^*), \mathbf{s}' = (s_1', \dots, s_n')</math>.</p>
---

**Algorithm 2:** Split: Splitting a vector in  $\mathbb{B}$ .

further show, that the expected value of  $r^*$  vanishes for  $\delta \rightarrow \infty$ .<sup>2</sup> These lemmas imply that  $\mathbf{r}_0$  and  $\mathbf{r}_1$  do not contain any information about  $\mathbf{s}$ .

**Lemma 1.** *If  $\mathbf{s}$  is uniformly distributed in  $\mathbb{B}(N)$ , the outputs  $\mathbf{r}_0$  and  $\mathbf{r}_1$  of Algorithm 1 are uniformly distributed in  $\delta\mathbb{B}(N)$ .*

**Lemma 2.** *Let  $(\mathbf{s}^*, \mathbf{s}') \leftarrow \text{Split}(\mathbf{s}, k)$  where  $\mathbf{s} \leftarrow \mathbb{B}(N)$  uniform and let  $(\mathbf{r}_0, \mathbf{r}_1, \mathbf{r}^*) \leftarrow \text{ShrVec}_\delta(\mathbf{s}')$ . Then, the outputs  $\mathbf{r}_0$  and  $\mathbf{r}_1$  of Algorithm 1 are uniformly distributed in  $\delta\mathbb{B}(N)$ .*

## 5 Delegation Algorithms

In this section, we present two delegation algorithms and their implementation under different assumptions. In both algorithms we want to delegate the computation of  $\mathfrak{s} * E$  from  $(\mathfrak{s}, E)$ . The first algorithm, **CIsO** keeps  $\mathfrak{s}$  hidden from the servers, while the second algorithm **HIsO**, keeps  $\mathfrak{s}$  and  $\mathfrak{s} * E$  hidden from the servers. For the efficiency reasons discussed in [7], we assume that there is a short representation  $\mathfrak{s} = (s_1, \dots, s_n) \in \mathbb{B}(N)$  of  $\mathfrak{s} = \prod_{i=1}^n \mathfrak{t}^{s_i}$ . In the case where  $\text{Cl}(\mathcal{O})$  is known, we further assume that  $s \in \mathbb{Z}_{\#\text{Cl}(\mathcal{O})}$  is known by the delegator, such that  $\mathfrak{s} = \mathfrak{g}^s$ . We define the two algorithms below, using the formalism from [15].

**Definition 5 (CIsO and HIsO).** *The isogeny computation algorithm **CIsO** and the hidden isogeny computation algorithm **HIsO** take as inputs a supersingular elliptic curve  $E/\mathbb{F}_p$  and an ideal  $\mathfrak{s}$ , either as an element in  $\mathbb{Z}_{\#\text{Cl}(\mathcal{O})}$  or a vector in  $\mathbb{B}(N)$ , then return the elliptic curve  $\mathfrak{s} * E$ . The input  $E$  is (honest/adversarial) unprotected, while  $\mathfrak{s}$  is secret or (honest/adversarial) protected. The output  $\mathfrak{s} * E$  of **CIsO** is unprotected, while it is protected in the case of **HIsO**. We write*

$$\mathfrak{s} * E \leftarrow \text{CIsO}(\mathfrak{s}, E) \quad \text{and} \quad \mathfrak{s} * E \leftarrow \text{HIsO}(\mathfrak{s}, E).$$

Below, we present implementations for both **CIsO** and **HIsO** in the OMTUP and 2HBC assumptions (Definitions 3 and 4). Both work in two rounds of delegation.

<sup>2</sup> This also follows intuitively from the fact, that the interval  $[-(\delta + 1)B, -\delta B - 1] \cup [\delta B + 1, (\delta + 1)B]$  is constant in size, while  $[-\delta B, \delta B]$  grows with increasing  $\delta$ , making it less and less probable for  $r_0, r_1$  to be sampled from the former.

## 5.1 Clso: Unprotected Codomain

Our general approach to hide  $\mathfrak{s}$  from the servers is to split it up into two ideals  $\mathfrak{a}_1, \mathfrak{a}_2$ , such that the consecutive application of both yields  $\mathfrak{a}_1 * (\mathfrak{a}_2 * E) = \mathfrak{a}_2 * (\mathfrak{a}_1 * E) = \mathfrak{s} * E$ , i.e. that we can compute the desired codomain in two rounds of delegation. In the 2HBC case, this can be implemented more or less straightforwardly. If, however, one of the servers is malicious, it could simply return a wrong codomain. Thus in the OMTUP case we want to be able to verify these computations. Unfortunately, unlike in the DLOG setting (e.g. see [15]), we can not compose elliptic curves in order to verify correctness, so we have to resort to comparisons, i.e. let two servers compute the same curve and check if they are the same. Note that simply going two different paths to  $\mathfrak{s} * E$  and comparing the results is also not possible, since the malicious server would take part in the computation of both of them and could simply apply another isogeny defined by an ideal  $\mathfrak{r}$  to its result in both rounds yielding the result  $\mathfrak{r} * (\mathfrak{s} * E)$  in both cases.

The goal of the verification is that the servers do not return an incorrect codomain without the delegator realizing (up to a certain probability). Note that we need to be able to verify intermediate results as well. We resort to direct comparisons, i.e. giving both servers common queries whose output we can directly compare. In the first round, we have the starting curve at our disposal, which easily allows to make the same queries to both servers. The second round becomes more tricky, however, since all the curves at our disposal are the starting curve and the curves generated by the servers in the first round, potentially maliciously. Reusing the starting curve in some queries while not in others makes the queries distinguishable. One obvious possibility would be to generate curves ourselves, which would however defeat the purpose of delegating in the first place. An alternative would be to work with lookup-tables analogous to the DLOG setting, but since we can not combine multiple elliptic curves, elements of the form  $(\mathfrak{a}, \mathfrak{a} * E)$  could only be used individually. Again, using such sets ends up defeating the need for delegation. Therefore our algorithm in the OMTUP case resorts to delegating sets of extra curves in order to increase verifiability.

To this end, we generate a set  $\mathcal{S}$  of ideal tuples  $(\mathfrak{c}_1, \mathfrak{c}_2, \mathfrak{d}_1, \mathfrak{d}_2)$  that satisfy  $\mathfrak{c}_1 \mathfrak{c}_2 = \mathfrak{d}_1 \mathfrak{d}_2$ . If we work over  $\mathbb{Z}_{\#Cl(\mathcal{O})}$ , this is straightforward. If we work with elements in  $\delta\mathbb{B}(N)$ , we can implement this as follows: for  $i = 1, \dots, n$ , generate  $c_{1,i}, c_{2,i}, d_{1,i} \leftarrow \chi(\delta_i B_i)$  and define  $d_{2,i} = c_{1,i} + c_{2,i} - d_{1,i}$  until  $d_{2,i} \in [-\delta_i B_i, \delta_i B_i]$ . Note that this approach might yield some information about  $c_{1,i} + c_{2,i}$  (at most that it is positive or negative) given  $c_{1,i}$  only, but we do not really need to care about that, since this is not enough information to be able to distinguish  $d_{2,i}$  from a random value (mainly because  $d_{1,i}$  remains unknown), so this will neither reduce the security nor the verifiability of the scheme. In the first round, we further delegate the computation of a second set  $\mathcal{R}$  of ideals applied to the starting curve and directly compare between the servers to increase verifiability.

We present our approach for the 2HBC assumption in Figure 1 and our approach for the OMTUP assumption in Figure 2. We analyze these protocols and discuss secure parameter sizes in Section 5.3. Note that the case (a) corresponds to the delegation algorithm presented in [15] with unprotected base element.

**Clso: 2HBC case.**

**Input** : Ideal  $\mathfrak{s}$ , elliptic curve  $E$ .  
**Output** : Elliptic curve  $\mathfrak{s} * E$  or  $\perp$ .

1. Generate the ideals  $\mathfrak{a}_1, \mathfrak{a}_2$  as follows.
  - (a) If  $\text{Cl}(\mathcal{O})$  is known, generate  $a_1 \leftarrow \mathbb{Z}_{\#\text{Cl}(\mathcal{O})}$  uniformly and compute  $a_2 = s - a_1$ .
  - (b) If  $\text{Cl}(\mathcal{O})$  is not known, generate  $(\mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{a}^*) \leftarrow \text{ShrVec}(s)$ .
2. Delegate the computation of  $E_1 \leftarrow \mathcal{U}_1((\mathfrak{a}_1, E))$ .
3. In the case where  $\text{Cl}(\mathcal{O})$  is not known, compute  $E_1 \leftarrow \mathfrak{a}^* * E_1$  locally.
4. Delegate the computation of  $E_2 \leftarrow \mathcal{U}_2((\mathfrak{a}_2, E_1))$ .
5. Return  $E_2$ .

**Fig. 1.** Implementation of **Clso** in the 2HBC assumption.

## 5.2 Hlso: Hidden Codomain

Next to keeping  $\mathfrak{s}$  hidden, **Hlso** also does not disclose the codomain curve to the auxiliary servers. The idea works similar to **Clso**, but rather than shrouding and delegating the computation of the isogeny generated by some secret ideal  $\mathfrak{s}$ , we do the same for an ideal  $\mathfrak{s}'$  to yield a codomain  $\mathfrak{s}' * E$  that can be known to the servers. The goal is to choose  $\mathfrak{s}'$ , so that  $\mathfrak{s}' * E$  is close enough to  $\mathfrak{s} * E$ , that the path can be efficiently computed by the delegator, while searching the space of potential curves is too large to reasonably allow an attacker to find  $\mathfrak{s} * E$  by walking from  $\mathfrak{s}' * E$ . We call the remaining path  $\mathfrak{s}^* = \mathfrak{s}\mathfrak{s}'^{-1}$ , so that  $\mathfrak{s}^* * (\mathfrak{s}' * E) = \mathfrak{s} * E$ .

To be able to assess path lengths, we work with ideals only in their vector representation in  $\mathbb{B}(N)$ . In the case where the class group  $\text{Cl}(\mathcal{O})$  is known, this is achieved by working modulo the relation lattice [3].<sup>3</sup> We then call  $\tilde{\mathbb{B}}(N) \subseteq \mathbb{B}(N)$  the subset from which  $\mathfrak{s}^*$  is sampled. We can achieve this splitting of  $\mathfrak{s}$  using the Split-procedure (Algorithm 2). The protocol then uses **Clso** as a subroutine with  $\mathfrak{s}'$  as the secret argument. It is summarized in Figure 3. Note that the protocol has the same description in the 2HBC and OMTUP assumptions, and that **Clso** is called with the appropriate assumption.

## 5.3 Analysis

**Size of  $k$ .** Assume we work with a class group of size approximately  $N$ , which has an associated quantum security level  $\lambda(N)$  with respect to GAIP (Definition 1). Let  $D = \#\mathbb{B}(N)$  denote the number of possible vectors in  $\mathbb{B}(N)$ . The basic idea is to define a subset  $\tilde{\mathbb{B}}(N) \subseteq \mathbb{B}(N)$  of size  $\tilde{D} = \#\tilde{\mathbb{B}}(N)$ , that is big enough that searching the entire space is at least as hard as breaking a GAIP instance.

<sup>3</sup> Note that  $\mathbb{B}(N)$  does not necessarily contain a representation for all elements in  $\text{Cl}(\mathcal{O})$ . We ignore this case and assume we can delegate such elements using simple heuristics, such as computing the “overshoot” locally, or simply by resampling.

**Clso: OMTUP case.**

**Input** : Ideal  $\mathfrak{s}$ , elliptic curve  $E$ .

**Output** : Elliptic curve  $\mathfrak{s} * E$  or  $\perp$ .

1. Generate the ideals  $\mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{b}_1, \mathfrak{b}_2$  as follows.
  - (a) If  $\text{Cl}(\mathcal{O})$  is known, generate two random elements  $a_1, b_1 \leftarrow \mathbb{Z}_{\#\text{Cl}(\mathcal{O})}$  and compute  $a_2 = s - a_1$  and  $b_2 = s - b_1$ .
  - (b) If  $\text{Cl}(\mathcal{O})$  is not known, generate  $(\mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{a}^*) \leftarrow \text{ShrVec}(s)$  and  $(\mathfrak{b}_1, \mathfrak{b}_2, \mathfrak{b}^*) \leftarrow \text{ShrVec}(s)$ .

Further, generate a set of random ideals  $\mathcal{R} = \{\mathfrak{c} \mid \mathfrak{c} \leftarrow \text{Cl}(\mathcal{O})\}$  and a set of random ideal tuples  $\mathcal{S} = \{(\mathfrak{c}_1, \mathfrak{c}_2, \mathfrak{d}_1, \mathfrak{d}_2) \leftarrow \text{Cl}(\mathcal{O})^4 \mid \mathfrak{c}_1 \mathfrak{c}_2 = \mathfrak{d}_1 \mathfrak{d}_2\}$ , where all the ideals are generated using  $\mathbb{Z}_{\#\text{Cl}(\mathcal{O})}$  or  $\delta\mathbb{B}(N)$ , respectively.

2. Delegate the computation of

$$E_{\mathfrak{a}_1}, \{E_{\mathfrak{c}_1}\}, \{E_{\mathfrak{c}}\} \leftarrow \mathcal{U}_1\left((\mathfrak{a}_1, E), \{(\mathfrak{c}_1, E) \mid \mathfrak{c}_1 \in \mathcal{S}\}, \{(\mathfrak{c}, E) \mid \mathfrak{c} \in \mathcal{R}\}\right),$$

$$E_{\mathfrak{b}_1}, \{E_{\mathfrak{d}_1}\}, \{E'_{\mathfrak{c}}\} \leftarrow \mathcal{U}_2\left((\mathfrak{b}_1, E), \{(\mathfrak{d}_1, E) \mid \mathfrak{d}_1 \in \mathcal{S}\}, \{(\mathfrak{c}, E) \mid \mathfrak{c} \in \mathcal{R}\}\right).$$

3. Verify if  $E_{\mathfrak{c}} \stackrel{?}{=} E'_{\mathfrak{c}}$  for  $\mathfrak{c} \in \mathcal{R}$ . If not, return  $\perp$ , otherwise continue.
4. In the case with  $\text{Cl}(\mathcal{O})$  unknown, locally compute

$$E_{\mathfrak{a}_1} \leftarrow \mathfrak{a}^* * E_{\mathfrak{a}_1}, \quad E_{\mathfrak{b}_1} \leftarrow \mathfrak{b}^* * E_{\mathfrak{b}_1}.$$

5. Delegate the computation of

$$E_s, \{E_{\mathfrak{d}}\} \leftarrow \mathcal{U}_1\left((\mathfrak{b}_2, E_{\mathfrak{b}_1}), \{(\mathfrak{d}_2, E_{\mathfrak{d}_1}) \mid \mathfrak{d}_2 \in \mathcal{S}\}\right),$$

$$E'_{\mathfrak{s}}, \{E_{\mathfrak{c}}\} \leftarrow \mathcal{U}_2\left((\mathfrak{a}_2, E_{\mathfrak{a}_1}), \{(\mathfrak{c}_2, E_{\mathfrak{c}_1}) \mid \mathfrak{c}_2 \in \mathcal{S}\}\right).$$

6. Verify if  $E_s \stackrel{?}{=} E'_{\mathfrak{s}}$  and if all  $E_{\mathfrak{d}} \stackrel{?}{=} E_{\mathfrak{c}}$ . If not, return  $\perp$ , otherwise return  $E_s$ .

**Fig. 2.** Implementation of **Clso** in the OMTUP assumption.

Since the servers are only given  $\mathfrak{s}' * E$ , they cannot resort to a meet-in-the-middle attack to find information about  $\mathfrak{s} * E$ , but rather have to resort to a database search of size  $\tilde{D}$  to find it. We assume that they would be able to identify the correct curve once found (e.g. by being able to decrypt a given ciphertext). The best known quantum algorithm for this database search is Grover's algorithm [14], which runs in  $O(\tilde{D}^{1/2})$ . Thus in order to ensure a quantum security level of  $\lambda$ , we choose  $\tilde{D} = 2^{2\lambda}$ . We can therefore define  $\tilde{\mathbb{B}}(N)$  analogously to  $\mathbb{B}(N)$ , i.e.  $\tilde{\mathbb{B}}(N) = \tilde{\mathbb{B}}_1 \times \dots \times \tilde{\mathbb{B}}_n$ , where  $\tilde{\mathbb{B}}_i \in \{[0, 0], \mathbb{B}_i\}$  of size  $\tilde{d}_i \in \{1, d_i\}$ , such that  $\tilde{D} = \prod_{i=1}^n \tilde{d}_i \approx 2^{2\lambda}$ .

The input parameter  $k$  of **Split** determines the number of non-zero  $\tilde{\mathbb{B}}_i$ . Thus, we need to choose  $k$  large enough such that an adversary's search space is approximately  $2^{2\lambda}$ . We note that due to Lemma 2, the adversary can not distinguish

**Hlso: General case****Input** : Ideal  $\mathfrak{s}$ , elliptic curve  $E$ , parameter  $k$ .**Output** : Elliptic curve  $\mathfrak{s} * E$  or  $\perp$ .

1. Compute  $(\mathfrak{s}^*, \mathfrak{s}') \leftarrow \text{Split}(\mathfrak{s}, k)$ .
2. Delegate  $E' \leftarrow \text{Clso}(\mathfrak{s}', E)$ .
3. Compute  $E_s = \mathfrak{s}^* * E'$  locally.
4. Return  $E_s$ .

**Fig. 3.** Implementation of **Hlso** for both 2HBC and OMTUP assumptions.

in which entries  $\mathfrak{s}'$  is zero and can therefore not know the subset  $C^*$ . Thus, the size of the search space can be determined by searching through any  $k$ -out-of- $n$  subsets and running through all permutations in these subsets. Therefore, we have to choose  $k$ , such that

$$\binom{n}{k} \prod_{i \in C^*} d_i \approx 2^{2\lambda}. \quad (1)$$

**Verifiability in the OMTUP Case.** In the OMTUP case, the servers successfully cheat if all of the verification conditions succeed but the output is wrong, i.e.  $E_s \neq \mathfrak{s} * E$ . Let us assume  $\mathcal{U}_1$  is the malicious server. In order to be successful,  $\mathcal{U}_1$  needs to correctly identify the query  $(\mathfrak{a}_1, E)$  in the first round and  $(\mathfrak{b}_2, E_{\mathfrak{b}_1})$  in the second round. Note that  $\mathcal{U}_1$  can also change the elements in  $\mathcal{S}$ , as long as it does so consistently in both rounds. The elements in  $\mathcal{R}$  have to be returned correctly, since they are directly compared to  $\mathcal{U}_2$ 's results.

Let  $m_s = \#\mathcal{S}$  and  $m_r = \#\mathcal{R}$ . By choosing a random subset of size  $\kappa \in \{1, \dots, 1+m_s\}$  among the queries of the first round, the probability of choosing a set that includes  $\mathfrak{a}_1$  (or  $\mathfrak{b}_1$ ) and no elements of  $\mathcal{R}$  is given by  $\binom{m_s}{\kappa-1} / \binom{1+m_s+m_r}{\kappa}$ . Furthermore, in the second round, the malicious server has to identify the *same* subset, which it achieves with probability  $1 / \binom{1+m_s}{\kappa}$ , yielding the full success probability for the adversary of

$$Pr[\text{success}] = \frac{\binom{m_s}{\kappa-1}}{\binom{1+m_s+m_r}{\kappa} \binom{1+m_s}{\kappa}} = \frac{\kappa}{1+m_s} \frac{\kappa!(m_s+m_r+1-\kappa)!}{(m_s+m_r+1)!}. \quad (2)$$

If  $m_r = 0, 1, 2$ , this probability is maximal for  $\kappa = 1 + m_s$ , while for  $m_r \geq 3$ , we find  $\kappa = 1$  to be optimal. In the latter case, the upper probability simplifies to  $Pr[\text{success} \mid m_r \geq 3] = [(1+m_s)(1+m_s+m_r)]^{-1}$ . Since this probability decreases quadratically with bigger  $m_s$ , we minimize the overall set sizes (and thus communication cost) by fixing  $m_r = 3$  and choosing  $m_s$  to yield the desired

verifiability. We thus find the verifiability

$$\beta(m_s) = 1 - \Pr[\text{success} \mid m_r = 3] = \frac{m_s^2 + 5m_s + 3}{m_s^2 + 5m_s + 4}. \quad (3)$$

**Security Proofs.** We prove security of **Clso** and **Hlso**.

**Theorem 1.** *Figure 1 is an outsource-secure implementation of **Clso** in the  $2HBC$  assumption.*

*Proof.* Correctness follows immediately from  $a_1 + a_2 = s$  or from the correctness of **ShrVec**, respectively. We prove security by proposing the following simulators:

- Environment  $\mathcal{E}$ : If  $\mathfrak{s}$  is not secret, both simulators behave as in the real execution of the protocol. Otherwise, in each round,  $\mathcal{S}_1$  generates random ideals  $\mathbf{u}_1, \mathbf{u}_2$  either as elements in  $\mathbb{Z}_{\#C1(\mathcal{O})}$  (case (a)) or as vectors in  $\delta\mathbb{B}(N)$  (case (b)). In the second case,  $\mathcal{S}_1$  further generates  $\mathbf{u}^* \leftarrow \mathbb{B}(N)$ . Then  $\mathcal{S}_1$  makes the query  $E_1 \leftarrow \mathcal{U}_1((\mathbf{u}_1, E))$ , computes  $E_1 \leftarrow \mathbf{u}^* * E_1$  if applicable, then makes the query  $E_2 \leftarrow \mathcal{U}_2((\mathbf{u}_2, E_1))$ .  $\mathcal{S}_1$  returns  $E_2$  and saves its own state and those of the servers. In any round, the input values  $\mathbf{u}_1, \mathbf{u}_2$  are indistinguishable from  $\mathbf{a}_1, \mathbf{a}_2$ . In case (b), this is given by Lemma 1.
- Servers  $\mathcal{U}_1, \mathcal{U}_2$ : For any  $\mathfrak{s}$ , the simulator  $\mathcal{S}_2$  proceeds exactly as the simulator  $\mathcal{S}_1$  for a secret  $\mathfrak{s}$ .  $\mathcal{UVIEW}_{\text{real}} \sim \mathcal{UVIEW}_{\text{ideal}}$  is guaranteed by the indistinguishability of  $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}^*$  and  $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}^*$ . Note that applying  $\mathbf{a}^* * E_1$  between the two queries has the advantage that neither server will see both the domain and the codomain of this isogeny and therefore cannot recover  $\mathbf{a}^*$ .  $\square$

**Theorem 2.** *Figure 2 is an outsource-secure implementation of **Clso** in the  $OMTUP$  assumption.*

*Proof.* Correctness of the output follows again from the definition of  $\mathfrak{s}$ . Concerning the verification conditions, correctness of  $E_c \stackrel{?}{=} E_d$  follows from the definition of  $\mathcal{S}$ . The other verification conditions are simple comparison operations between both servers. We prove security by proposing the following simulators:

- Environment  $\mathcal{E}$ : If  $\mathfrak{s}$  is not secret, both simulators behave as in the real execution of the protocol. Otherwise, in each round,  $\mathcal{S}_1$  generates random ideals  $\mathbf{u}_1, \mathbf{u}_2, \mathbf{v}_1, \mathbf{v}_2$  and in case (b) further  $\mathbf{u}^*, \mathbf{v}^*$  as vectors in  $\mathbb{B}(N)$ .  $\mathcal{S}_1$  further generates two random sets of ideals  $\mathcal{M}_1, \mathcal{M}_2$  of size  $m_r$  and four sets of ideals  $\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3, \mathcal{N}_4$  of size  $m_s$ , such that for  $(\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3, \mathbf{n}_4)_i \in \mathcal{N}_1 \times \mathcal{N}_2 \times \mathcal{N}_3 \times \mathcal{N}_4$ , it holds that  $\mathbf{n}_1 \mathbf{n}_4 = \mathbf{n}_2 \mathbf{n}_3$ , pairwise for  $i = 1, \dots, m_s$ . Then  $\mathcal{S}_1$  makes the queries

$$E_{\mathbf{u}_1}, \{E_{\mathbf{n}_1}\}, \{E_{\mathbf{m}_1}\} \leftarrow \mathcal{U}_1\left((\mathbf{u}_1, E), \{(\mathbf{n}_1, E) \mid \mathbf{n}_1 \in \mathcal{N}_1\}, \{(\mathbf{m}_1, E) \mid \mathbf{m}_1 \in \mathcal{M}_1\}\right),$$

$$E_{\mathbf{v}_1}, \{E_{\mathbf{n}_2}\}, \{E_{\mathbf{m}_2}\} \leftarrow \mathcal{U}_2\left((\mathbf{v}_1, E), \{(\mathbf{n}_2, E) \mid \mathbf{n}_2 \in \mathcal{N}_2\}, \{(\mathbf{m}_2, E) \mid \mathbf{m}_2 \in \mathcal{M}_2\}\right).$$

$\mathcal{S}_1$  verifies the results. If either of the elements in  $\{E_{\mathbf{m}_1}\}$  or  $\{E_{\mathbf{m}_2}\}$  are incorrect, then  $\mathcal{S}_1$  returns  $\perp$ , otherwise it continues. In case (b),  $\mathcal{S}_1$  computes

$E_{u_1} \leftarrow \mathbf{u}^* * E_{u_1}$  and  $E_{v_1} \leftarrow \mathbf{v}^* * E_{v_1}$ . Then, in the second round,  $\mathcal{S}_1$  makes the queries

$$E_{v_2}, \{E_{n_3}\}, \leftarrow \mathcal{U}_1 \left( (\mathbf{v}_2, E_{v_1}), \{(\mathbf{n}_3, E_{n_2}) \mid \mathbf{n}_3 \in \mathcal{N}_3\} \right),$$

$$E_{u_2}, \{E_{n_4}\}, \leftarrow \mathcal{U}_2 \left( (\mathbf{u}_2, E_{u_1}), \{(\mathbf{n}_4, E_{n_1}) \mid \mathbf{n}_4 \in \mathcal{N}_4\} \right).$$

Again,  $\mathcal{S}_1$  verifies the results. If  $\nexists \mathbf{x} : E_{u_2} = (\mathbf{x}u_1u_2) * E \wedge E_{v_2} = (\mathbf{x}v_1v_2) * E$ ,  $\mathcal{S}_1$  returns  $\perp$ . Otherwise, let  $\kappa$  be the number of pairs  $(E_{n_3}, E_{n_4})$  for which there doesn't exist such an  $\mathbf{x}$ . Then with probability  $1 - Pr[success]$  (as given in equation (2)),  $\mathcal{S}_1$  returns  $E_s$ , otherwise  $\mathcal{S}_1$  returns  $\perp$ .  $\mathcal{S}_1$  saves the appropriate states. In any round of the simulation, the input tuple  $(\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}^*, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}^*, \mathcal{M}_1, \mathcal{M}_2, \mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3, \mathcal{N}_4)$  is indistinguishable from the tuple  $(\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}^*, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}^*, \mathcal{R}, \mathcal{R}, \{\mathbf{c}_1 \in \mathcal{S}\}, \{\mathbf{d}_1 \in \mathcal{S}\}, \{\mathbf{d}_2 \in \mathcal{S}\}, \{\mathbf{c}_2 \in \mathcal{S}\})$ , due to uniform sampling or because of Lemma 1. If a server cheats,  $\mathcal{S}_1$  outputs a wrong result with probability  $Pr[success]$ , otherwise it returns  $\perp$ , as in the real execution of the protocol. It follows  $\mathcal{E}\mathcal{V}\mathcal{I}\mathcal{E}\mathcal{W}_{\text{real}} \sim \mathcal{E}\mathcal{V}\mathcal{I}\mathcal{E}\mathcal{W}_{\text{ideal}}$ .

- Servers  $\mathcal{U}_1, \mathcal{U}_2$ : For any  $\mathfrak{s}$ , the simulator  $\mathcal{S}_2$  proceeds exactly as the simulator  $\mathcal{S}_1$  for a secret  $\mathfrak{s}$ , except for the verification procedure after the second round, which is not necessary.  $\mathcal{U}\mathcal{V}\mathcal{I}\mathcal{E}\mathcal{W}_{\text{real}} \sim \mathcal{U}\mathcal{V}\mathcal{I}\mathcal{E}\mathcal{W}_{\text{ideal}}$  is guaranteed by the indistinguishability of the tuple described above.  $\square$

**Theorem 3.** *Figure 3 is an outsource-secure implementation of Hlso in both the 2HBC and OMTUP assumptions.*

*Proof.* Correctness of the output follows from the correctness of Split and Clso. Security follows from the outsource-security of Clso and the appropriate choice of the parameter  $k$  as determined by equation (1).

*Remark 1.* Note that Definition 4 implies that  $\mathcal{U}_1$  and  $\mathcal{U}_2$  might try to collude. Yet, since their outputs are honestly generated, their indirect communication channel through  $\mathcal{T}$  is in fact non-existent. For example,  $E_1$ , output by  $\mathcal{U}_1$  and input to  $\mathcal{U}_2$ , is honestly generated and can therefore not contain any auxiliary information that  $\mathcal{U}_2$  could use to learn any information about  $\mathbf{a}_1$ . Definition 3 implies that at least one of the two servers is honest, so that collusion is not possible in the OMTUP case.

**Communication Cost.** We want to express the communication cost between the delegator and the server. We do this by looking at the information content of the exchanged elements in bits. We establish the following maximal costs.

Element of	Maximal cost in bits
$\mathbb{Z}_{\#\text{Cl}(\mathcal{O})}$	$\lceil \log_2 \#\text{Cl}(\mathcal{O}) \rceil$
$\delta\mathbb{B}(N)$	$\sum_{i=1}^n \log_2(2\delta_i B_i + 1)$
$\mathbb{F}_p$	$\lceil \log_2 p \rceil$

Note that elliptic curves in Montgomery form are encoded by a single curve parameter in  $\mathbb{F}_p$ . The actual average communication cost of elements in  $\mathbb{B}$  and

$\delta\mathbb{B}$  is smaller than the maximal cost if the individual vector entries are expressed using the minimal amount of bits. This representation considerably lowers the communication cost, especially for large  $\delta_i$ . We can estimate the communication costs by establishing the minimal number of bits of an element uniformly sampled from  $\delta\mathbb{B}$  as

$$\text{Ex}_I(\delta\mathbb{B}) := \sum_{i=1}^n \frac{1}{2\delta_i B_i + 1} \sum_{y=-\delta_i B_i}^{\delta_i B_i} \lceil \log_2(2|y| + 1) \rceil.$$

We can now establish the communication cost for the delegation of **Clso** and **Hlso**, which are the same. In the 2HBC case, the delegator uploads one element from either  $\mathbb{Z}_{\#\text{Cl}(\mathcal{O})}$  or  $\delta\mathbb{B}(N)$  and downloads one elliptic curve from each server, defined by a parameter in  $\mathbb{F}_p$ . In the OMTUP case, the delegator uploads  $2 + 2m_s + m_r$  elements from either  $\mathbb{Z}_{\#\text{Cl}(\mathcal{O})}$  or  $\delta\mathbb{B}(N)$  to each server and downloads the same amount of elliptic curves. We define the upload and download costs per server in the 2HBC ( $b = 0$ ) and OMTUP ( $b = 1$ ) case:

$$\begin{aligned} \text{Up}(x) &= \begin{cases} (1 + (2m_s + m_r + 1)b) \lceil \log_2 \#\text{Cl}(\mathcal{O}) \rceil, & x = \text{Cl}(\mathcal{O}), \\ (1 + (2m_s + m_r + 1)b) \text{Ex}_I(\delta\mathbb{B}), & x = \delta\mathbb{B}, \end{cases} \quad (4) \\ \text{Down} &= (1 + (2m_s + m_r + 1)b) \lceil \log_2 p \rceil, \end{aligned}$$

**Cost Reduction Functions.** Ignoring the costs of comparison operations, element generation and **ShrVec**, as they are negligible in comparison to isogeny computations, we get the following cost reduction functions for **Clso** and **Hlso**

$$\alpha_{\text{Clso}}(\delta, B, n, b) = \frac{(1+b)\mathcal{I}(r^*)}{\mathcal{I}(s)}, \quad \alpha_{\text{Hlso}}(\delta, B, n, k, b) = \frac{(1+b)\mathcal{I}(r^*) + \mathcal{I}(s^*)}{\mathcal{I}(s)}, \quad (5)$$

where the parameter  $b \in \{0, 1\}$  distinguishes between the 2HBC and OMTUP cases, respectively, and where  $s, r^*, s^*$  and  $r$  relate to the outputs of **Split** and **ShrVec**. Note that the isogeny cost functions all depend on  $B$  and  $n$ . Further, the size of  $r^*$  (thus  $\mathcal{I}(r^*)$ ) depends on  $\delta$  and the size of  $s^*$  (thus  $\mathcal{I}(s^*)$ ) on  $k$ . Remember that in the case, where the class group and relation lattice are known, we do not need to use **ShrVec**, so that, effectively,  $\mathcal{I}(r^*) = 0$ . Similarly, since the expected value of  $r^*$  vanishes for  $\delta \rightarrow \infty$ , we can identify

$$\begin{aligned} \alpha_{\text{Clso}}^{\text{Cl}(\mathcal{O})}(B, n) &= \lim_{\delta \rightarrow \infty} \alpha_{\text{Clso}}(\delta, B, n, b) = 0, \\ \alpha_{\text{Hlso}}^{\text{Cl}(\mathcal{O})}(B, n, k) &= \lim_{\delta \rightarrow \infty} \alpha_{\text{Hlso}}(\delta, B, n, k, b) = \frac{\mathcal{I}(s^*)}{\mathcal{I}(s)}. \end{aligned} \quad (6)$$

In this case the cost also becomes independent of  $b$ , i.e. of the underlying server assumption. Each server, on the other hand, has to compute  $(2m_s + m_r + 1)b + 1$  isogenies of cost given by  $\mathcal{I}(r)$ . We therefore find the relative cost per server of

$$\alpha_{\mathcal{U}}(\delta, B, n, b) = \frac{((2m_s + m_r + 1)b + 1)\mathcal{I}(r)}{\mathcal{I}(s)}$$

where the size of  $r$  also depends on  $\delta$ . Note that we generally have  $\alpha_{\mathcal{U}}(\delta, B, n, b) \geq 1$ . We will still refer to this as the cost reduction function.

## 6 Applications

In this section we discuss how to apply our delegation algorithm to some of the isogeny-based protocols in the CSIDH setting and benchmark our delegation algorithms using the *VeluSqrt* implementation in MAGMA,<sup>4</sup> introduced in [2]. All benchmarks were done in Magma v2.25-6 on an Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz with 128 GB memory.<sup>5</sup> We note that our benchmarks support our assumption that ShrVec is negligible as its cost generally constitutes less than 0.01% of the cost of the delegator in terms of CPU cycles.

### 6.1 Delegating the CSIDH Key Exchange Protocol

We briefly revisit the CSIDH key exchange protocol in this section and then show how to delegate it. CSIDH uses a prime  $p = 4 \prod_{i=1}^n \ell_i - 1$  of appropriate size and defines the starting curve as  $E_0 : y^2 = x^3 + x$  over  $\mathbb{F}_p$ . Further, CSIDH uses symmetric boxes around 0, all of equal size, i.e.  $\mathbb{B}(N) = [-B, B]^n$ .

- *Key generation*: Alice’s private key is a vector  $\mathfrak{s} \in \mathbb{B}(N)$  representing  $\mathfrak{s}$  and her public key is  $E_A = \mathfrak{s} * E_0$ .
- *Key exchange*: Using Bob’s public key  $E_B$ , Alice can compute the shared secret  $\mathfrak{s} * E_B$ .

In terms of the input/output specifications from Definition 2, we consider  $\mathfrak{s}$  as a secret input,  $\mathfrak{s} * E_0$  as an unprotected output, and  $\mathfrak{s} * E_B$  as a secret or protected output. Note that we have to consider  $E_B$  as honestly generated, which can always be achieved by authenticating the public key. We can then use `CIso` to delegate the key generation step and `HIso` for the key exchange step as follows:

- *Key generation*: Delegate  $E_A \leftarrow \text{CIso}(\mathfrak{s}, E_0)$ .
- *Key exchange*: Delegate  $\mathfrak{s} * E_B \leftarrow \text{HIso}(\mathfrak{s}, E_B)$ .

We can easily see that the cost reduction function for the delegation of CSIDH can be expressed as  $\alpha_{\text{CSIDH}}(\delta, B, n, k, b) = \frac{1}{2}(\alpha_{\text{CIso}}(\delta, B, n, b) + \alpha_{\text{HIso}}(\delta, B, n, k, b))$ , while for the server, we have  $\alpha_{\mathcal{U}, \text{CSIDH}}(\delta, B, n, b) = \alpha_{\mathcal{U}}(\delta, B, n, b)$ .

*Instantiations.* We look at specific instantiations of CSIDH. While the security is still subject of scrutiny, we go on a limb and make certain assumptions in this section, which the reader should take with caution. Our estimates for  $\lambda$  are mainly based on the results in [4, Table 8], [9, Table 3] and [22, Figure 1].

**CSIDH-512.** The original proposal from [7] uses the following parameters:  $n = 74$ ,  $\log_2 p \approx 512$ ,  $B = 5$ , so that  $D = \#\mathbb{B}(N) = (2B + 1)^{74} \approx 2^{256}$ . For the key exchange round, we have to define  $k$  such that equation (1) is fulfilled. Looking at the different security assessments found in the literature, we take the

<sup>4</sup> <https://velusqrt.isogeny.org/software.html>

<sup>5</sup> Our implementation can be found here: <https://github.com/gemeis/DeCSIDH>

lower estimate of  $\lambda \approx 58$  from [22, Figure 1], which corresponds to  $k = 18$ . The benchmark results are summarized in Table 1.

**Table 1.** Benchmarks for CSIDH-512: In the left table are the benchmarked cost reduction functions for the delegator, while the right table shows the relative cost of the server. For the latter, we chose  $m_r = 3$  and compare the cases  $m_s = 0$  ( $\beta = 75\%$ ) and  $m_s = 8$  ( $\beta = 99\%$ ). Different  $m_r$  and  $m_s$  do not impact the delegator cost. Note that the case for  $\delta \rightarrow \infty$  corresponds to the cost of delegating CSIDH if the class group structure and relation lattice are known (cf. eq. (6)). For the CSIDH-512 parameter set this is indeed the case as the class group has been computed in [3].

$\alpha_{\text{CSIDH}}$	$\delta = 1$	5	10	100	$\rightarrow \infty$	$\alpha_{U, \text{CSIDH}}$	$\delta = 1$	5	10	100
2HBC	0.462	0.253	0.213	0.134	0.113	2HBC	0.971	4.59	8.84	91.9
OMTUP	0.877	0.391	0.322	0.159	0.113	OMTUP ( $m_s = 0$ )	4.83	20.5	42.1	395
						OMTUP ( $m_s = 8$ )	19.1	80.5	170	1376

*Communication Cost.* The communication cost of the full protocol is given by four times the costs established in equation (4), since  $\text{Clso}$  is invoked twice with two servers each time. The total costs are summarized in Table 2. The OMTUP case is strongly dependent on  $m_r$ ,  $m_s$ . But even if we want high verifiability and low cost in the OMTUP case, the communication cost is manageable, e.g. assuming  $m_r = 3$  and setting  $\delta = 100$  and  $m_s = 100$ , we find  $33kB$  of upload and  $13kB$  of download.

**Table 2.** Communication costs of CSIDH-512 in the 2HBC and OMTUP assumptions. In the OMTUP case, we choose  $m_r = 3$  and compare the cases  $m_s = 0$  ( $\beta = 75\%$ ) and  $m_s = 8$  ( $\beta = 99\%$ ). We compare different values of  $\delta$  and the case where the class group and relation lattice are known.

	$\text{Cl}(\mathcal{O})$	Upload				Download
		$\delta = 1$	5	10	100	
2HBC	129 B	108 B	180 B	215 B	333 B	256 B
OMTUP ( $m_s = 0$ )	645 B	539 B	900 B	1074 B	1663 B	1280 B
OMTUP ( $m_s = 8$ )	2.63 kB	2.21 kB	3.69 kB	4.40 kB	6.82 kB	5.25 kB

**CSIDH-1792 and SQALED CSIDH-4096.** As a comparison to CSIDH-512, we also consider the larger parameter set for CSIDH-1792 proposed and analyzed in [4] as well as the SQALED CSIDH-4096 proposal from [9]. The former has  $\log_2 p \approx 1792$ ,  $n = 209$ ,  $B = 10$ , and we find  $k = 24$  taking the value  $\lambda = 104$  from [4, Table 8]. CSIDH-4096 uses  $n = 417$ ,  $\log_2 p \approx 4096$  and  $B = 1$ , such that  $\#\mathbb{B}(N) \approx 2^{661} \ll \#\text{Cl}(\mathcal{O})$ . Using  $\lambda = 124$  as an estimate (cf. [9, Table 3]) yields

$k = 40$ . The results are summarized in Table 3. It is interesting to note that the gains in CSIDH-4096 are not considerably larger than for CSIDH-1792. This is mainly due to the fact, that the authors of [9] chose a key set that covers only a subset of the class group, such that the relative cost of local computations is lower than if the full group would be covered, resulting in a lower overall gain for the delegator.

**Table 3.** Benchmarked cost reduction functions for different  $\delta$ , representing CSIDH-1792 from [4] and SQALE'd CSIDH-4096 from [9], respectively, in the 2HBC and OMTUP assumptions. The case  $\delta \rightarrow \infty$  again represents the cost in case class group structure and relation lattice are known.

$\alpha_{\text{CSIDH}}^{1792}$	$\delta = 1$	5	10	100	$\rightarrow \infty$	$\alpha_{\text{CSIDH}}^{4096}$	$\delta = 1$	5	10	100	$\rightarrow \infty$
2HBC	0.331	0.123	0.103	0.067	0.042	2HBC	0.312	0.101	0.076	0.055	0.033
OMTUP	0.614	0.209	0.165	0.085	0.043	OMTUP	0.577	0.179	0.132	0.079	0.036

## 6.2 Signature Protocols

**SeaSign.** SeaSign is a signature protocol based on Fiat-Shamir with aborts [13] for cases where the class group is unknown. During the signature process, the signer needs to compute  $t$  isogenies  $\mathbf{b}_1, \dots, \mathbf{b}_t$  as commitments, where  $t$  is a security parameter that depends amongst others on the public key size  $2^s$ . Secure instantiations require  $st \geq \lambda$ . The exponents  $\mathbf{b}_i$  that define these isogenies are sampled from  $\mathbb{B}(N) = [-(nt+1)B, (nt+1)B]^n$  in order to guarantee a reasonable success probability. Further steps are the typical hashing and response computation, which we assume to have negligible cost. The verification has the same average computational cost as the signing process, as the commitments are verified using response vectors in  $\mathbb{B}(N)$ . Delegation can be achieved by using  $t$  instances of  $\text{Clso}$  (possibly in parallel). The delegator is left with computing the  $\mathbf{r}^*$ -part of each of these delegations, we therefore find

$$\alpha_{\text{SeaSign}}(\delta, B, n, t, b) = \frac{(1+b)\mathcal{I}(r^*)}{\mathcal{I}(s)},$$

choosing the same  $\delta$  for each step. We note that while  $r^*$  and  $s$  are sampled from the larger set  $\mathbb{B}(N) = [-(nt+1)B, (nt+1)B]^n$ , the cost difference between different  $t \in \{1, \dots, 128\}$  is negligible, so that we find  $\alpha_{\text{SeaSign}}(\delta, B, n, t, b) \approx \alpha_{\text{Clso}}(\delta, B, n, b)$ . The instantiation in [13] uses the parameter set from CSIDH-512 [7]. We show the cost reduction for different values of  $\delta$  in the top of Table 4. Because of the size of the set  $\mathbb{B}$ , the communication costs of delegating SeaSign become more expensive. In the OMTUP case, since we repeat the protocol throughout many rounds, we choose  $m_r = 3$  and  $m_s = 0$  for our assessment of the communication costs, which are summarized in the bottom of Table 4.

**CSI-FiSh.** One the main results of the CSI-FiSh paper [3] is the computation of the class group structure and relation lattice for the CSIDH-512 parameter set. Using the knowledge of  $\text{Cl}(\mathcal{O})$ , the authors construct a signature scheme in the random oracle model based on the original identification protocol from Rostovtsev and Stolbunov [24, 27]. The main computational effort of the signature process comes, analogous to SeaSign, from the fact that the signer needs to compute  $t$  isogenies given by  $\mathbf{b}_1, \dots, \mathbf{b}_t$ , depending on the public key size  $2^s$ . In contrast to SeaSign however, these elements can simply be sampled from  $\mathbb{Z}_{\#\text{Cl}(\mathcal{O})}$  and then translated into short vectors using the relation lattice. A verifier has to compute the same amount of isogenies and therefore has the same computational cost as the signer. Both the prover and verifier can delegate these isogenies using  $\text{Clso}$ , but knowing  $\text{Cl}(\mathcal{O})$  has now the advantage of not having to resort to  $\text{ShrVec}$ , and therefore not needing to compute the  $\tau^*$  part of the isogeny. This means that from the point of view of the delegator, the signature and its verification are basically free, up to element generation in  $\mathbb{Z}_{\#\text{Cl}(\mathcal{O})}$  and comparison operations. The communication costs for CSI-FiSh, again assuming  $m_r = 3$  and  $m_s = 0$  amount to  $64.25t$  bytes upload and  $128t$  bytes download in the 2HBC case and  $321.25t$  bytes upload and  $640t$  bytes download in the OMTUP case.

**Table 4.** Top: Benchmarked cost reduction function for different  $\delta$ . Bottom: Communication cost (assuming unknown  $\text{Cl}(\mathcal{O})$ ) in the 2HBC and OMTUP assumptions. We compare the cases  $t = 32$  and  $t = 128$ .

$\alpha_{\text{SeaSign}}$	$\delta = 1$	5	10	100	$\rightarrow \infty$
2HBC	0.393	0.162	0.120	0.031	0.003
OMTUP	0.809	0.315	0.226	0.057	0.003

  

	Upload				Download
	$\delta = 1$	5	10	100	
2HBC, $t = 32$	7.87 kB	9.19 kB	9.77 kB	11.7 kB	4.0 kB
2HBC, $t = 128$	36.1 kB	41.4 kB	43.7 kB	51.4 kB	16.0 kB
OMTUP, $t = 32$	39.4 kB	45.9 kB	48.8 kB	58.5 kB	20.0 kB
OMTUP, $t = 128$	181 kB	207 kB	218 kB	257 kB	80.0 kB

## 7 Conclusion

This work presents a first approach of securely and verifiably delegating isogeny computations to potentially untrusted servers in the CSIDH setting. We presented two algorithms and showed their application to different instances of CSIDH [7, 4, 9] and to the signature schemes SeaSign [13] and CSI-FiSh [3]. Our algorithms present a communication-cost trade-off. In terms of the cost reduction function, we reduced the delegator’s cost asymptotically (for large communication cost) down to 11.3% and about 3.5% of the cost of the local computation for

CSIDH-512 and SQALE'd CSIDH-4096, respectively, while the cost of SeaSign reduces to a few percent and asymptotically vanishes. Using the known class group of CSI-FiSh, its delegated cost reduces to element generation in  $\mathbb{Z}_{\#Cl(\mathcal{O})}$ .

Our protocols work in two rounds of delegation and use either the OMTUP or the 2HBC server assumptions. It is of interest to try to reduce delegation to a single round. The tools developed in this work do not seem to allow delegation to only malicious servers. We therefore leave it open to develop delegation schemes that work in the *two untrusted* or *one untrusted program model* presented in [15].

We also leave it as an open question to apply delegation to other post-quantum cryptographic paradigms, such as lattice-based and code-based cryptography.

## References

1. Azarderakhsh, R., Campagna, M., Costello, C., De Feo, L., Hess, B., Hutchinson, A., Jalali, A., Jao, D., Karabina, K., Koziel, B., LaMacchia, B., Longa, P., Naehrig, M., Pereira, G., Renes, J., Soukharev, V., Urbanik, D.: Supersingular isogeny key encapsulation. Submission to the NIST Post-Quantum Standardization project (2017)
2. Bernstein, D., De Feo, L., Leroux, A., Smith, B.: Faster computation of isogenies of large prime degree. arXiv preprint arXiv:2003.10118 (2020)
3. Beullens, W., Kleinjung, T., Vercauteren, F.: CSI-FiSh: efficient isogeny based signatures through class group computations. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 227–247. Springer (2019)
4. Bonnetain, X., Schrottenloher, A.: Quantum security analysis of CSIDH. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 493–522. Springer (2020)
5. Castryck, W., Decru, T.: CSIDH on the surface. In: International Conference on Post-Quantum Cryptography. pp. 111–129. Springer (2020)
6. Castryck, W., Galbraith, S.D., Farashahi, R.R.: Efficient arithmetic on elliptic curves using a mixed Edwards-Montgomery representation. IACR Cryptol. ePrint Arch. **2008**, 218 (2008)
7. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: an efficient post-quantum commutative group action. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 395–427. Springer (2018)
8. Charles, D.X., Lauter, K.E., Goren, E.Z.: Cryptographic hash functions from expander graphs. Journal of Cryptology **22**(1), 93–113 (2009)
9. Chávez-Saab, J., Chi-Dominguez, J.J., Jaques, S., Rodriguez-Henriquez, F.: The SQALE of CSIDH: Square-root Vélu quantum-resistant isogeny action with low exponents. Tech. rep., Cryptology ePrint Archive, Report 2020/1520 (2020)
10. Chevalier, C., Laguillaumie, F., Vergnaud, D.: Privately outsourcing exponentiation to a single server: cryptanalysis and optimal constructions. In: European Symposium on Research in Computer Security. pp. 261–278. Springer (2016)
11. Childs, A., Jao, D., Soukharev, V.: Constructing elliptic curve isogenies in quantum subexponential time. Journal of Mathematical Cryptology **8**(1), 1–29 (2014)
12. Couveignes, J.M.: Hard homogeneous spaces. IACR Cryptol. ePrint Arch. **2006**, 291 (2006)

13. De Feo, L., Galbraith, S.D.: SeaSign: Compact isogeny signatures from class group actions. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 759–789. Springer (2019)
14. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing. pp. 212–219 (1996)
15. Hohenberger, S., Lysyanskaya, A.: How to securely outsource cryptographic computations. In: Theory of Cryptography Conference. pp. 264–282. Springer (2005)
16. Jao, D., De Feo, L.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: International Workshop on Post-Quantum Cryptography. pp. 19–34. Springer (2011)
17. Kleinjung, T.: Quadratic sieving. *Mathematics of Computation* **85**(300), 1861–1873 (2016)
18. Kuperberg, G.: A subexponential-time quantum algorithm for the dihedral hidden subgroup problem. *SIAM Journal on Computing* **35**(1), 170–188 (2005)
19. NIST: NIST post-quantum cryptography, round 3 submissions (2020), <https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions>
20. Pedersen, R., Uzunkol, O.: Secure delegation of isogeny computations and cryptographic applications. In: Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop. pp. 29–42 (2019)
21. Pedersen, R., Uzunkol, O.: Delegating supersingular isogenies over  $\mathbb{F}_{p^2}$  with cryptographic applications. *IACR Cryptol. ePrint Arch.* **2021**, 506 (2021)
22. Peikert, C.: He gives C-sieves on the CSIDH. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 463–492. Springer (2020)
23. Regev, O.: A subexponential time algorithm for the dihedral hidden subgroup problem with polynomial space. arXiv preprint quant-ph/0406151 (2004)
24. Rostovtsev, A., Stolbunov, A.: Public-key cryptosystem based on isogenies. *IACR Cryptol. ePrint Arch.* **2006**, 145 (2006)
25. Siegel, C.: Über die classenzahl quadratischer zahlkörper. *Acta Arithmetica* **1**(1), 83–86 (1935)
26. SIKE: Supersingular Isogeny Key Encapsulation (2018), <https://sike.org>
27. Stolbunov, A.: Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves. *Advances in Mathematics of Communications* **4**(2), 215 (2010)
28. Stolbunov, A.: Cryptographic schemes based on isogenies. Norges teknisk-naturvitenskapelige universitet (2012)
29. Uzunkol, O., Rangasamy, J., Kuppusamy, L.: Hide the modulus: a secure non-interactive fully verifiable delegation scheme for modular exponentiations via CRT. In: International Conference on Information Security. pp. 250–267. Springer (2018)
30. Vélou, J.: Isogénies entre courbes elliptiques. *CR Acad. Sci. Paris, Séries A* **273**, 305–347 (1971)

## A Proving Lemmas 1 and 2

**Distributions and Expected Values of ShrVec and Split.** We first analyze the properties of Algorithm 1. Since they hold for any  $i \in \{1, \dots, n\}$ , we will omit the index. Correctness holds, since after the `repeat` loop, we have  $r_0 + r_1 = s$

and  $r^* = 0$ . If either of the following if-conditions succeed, then  $r_0 + r_1 = 0$  and  $r^* = s$ . In either case,  $r_0 + r_1 + r^* = s$  holds.

Before proving Lemma 1, we first introduce some more notation. We define the discrete rectangular function  $\Theta_x[x_{\text{start}}, x_{\text{end}}]$  which is 1, if  $x \in \{x_{\text{start}}, \dots, x_{\text{end}}\}$  and 0 otherwise. We also write  $f(x)|_{x_{\text{start}}}^{x_{\text{end}}} = f(x)\Theta_x[x_{\text{start}}, x_{\text{end}}]$  as a shorthand. For further conciseness in notation, we introduce

$$c_k = (\delta + k)B \quad , \quad \Delta_k = 2c_k + 1 \quad \text{and} \quad d = 2B + 1 .$$

In general, we denote the distribution of a value by the corresponding capital letter, e.g.  $S(x)$  represents the distribution of  $s$  etc. Finally, we write convolutions as  $f(x) * g(x) = \sum_{y=-\infty}^{\infty} f(y)g(x-y)$ . As an example for our notation, consider the trapezoidal distribution

$$\chi(c_0) \star \chi(B) = (d\Delta_0)^{-1} \left( (x + c_1 + 1)|_{-c_1}^{-c_{-1}-1} + d|_{-c_{-1}}^{c_{-1}} + (-x + c_1 + 1)|_{c_{-1}+1}^{c_1} \right) .$$

We further denote by  $H_n = \sum_{i=1}^n \frac{1}{i}$  the  $n$ -th harmonic number. We establish expected values for elements sampled from the distributions surrounding `ShrVec`. Since all of these distributions will turn out to be symmetric, we define the expected values in terms of the absolute values of the elements. The expected absolute value of an element from a distribution  $F(x)$  is thus

$$\text{Ex}_F := \sum_{y=-\infty}^{\infty} |y|F(y) .$$

As an example, consider the uniform distribution  $\chi(x)$  for which we find  $\text{Ex}_\chi(x) = \frac{x(x+1)}{2x+1}$ . This allows us to determine the expected values of elements from e.g.  $S(x)$  and  $R(x)$ :

$$\text{Ex}_S(B) := \text{Ex}_\chi(B) = \frac{1}{d}B(B+1) \quad \text{and} \quad \text{Ex}_R(\delta, B) := \text{Ex}_\chi(c_0) = \frac{1}{\Delta_0}c_0(c_0+1) \quad (7)$$

In order to prove Lemma 1, we analyze how the distribution of  $s$  and of  $r_0$  and  $r_1$  change throughout the algorithm. We define different instances of the distributions with different subscripts <sup>(i)</sup>.

1. We first analyze what happens in the `repeat`-loop. In order to fulfill the condition at the end of the loop, we distinguish two possible cases for  $r_0$ :
  - $r_0 \in [-c_0, c_0]$ : The `until`-condition always succeeds and we have

$$R_1^{(0)}(x) = \frac{\Theta_x[-c_0, c_0]}{\Delta_1} * \chi(B) = (d\Delta_1)^{-1} \begin{cases} x + c_1 + 1, & x \in [-c_1, -c_{-1}], \\ d, & x \in [-c_{-1}, c_{-1}], \\ -x + c_1 + 1, & x \in [c_{-1}, c_1]. \end{cases}$$

- $r_0 \in [-c_1, -c_0 - 1] \cup [c_0 + 1, c_1]$ : In this case, we have

$$R_1^{(0)}(x) = (d\Delta_1)^{-1} \begin{cases} -|x| + c_2 + 1 & x \in [-c_2, -c_1 - 1] \cup [c_1 + 1, c_2], \\ B & x \in [-c_1, -c_0 - 1] \cup [c_0 + 1, c_1], \\ |x| - c_{-1} & x \in [-c_0, -c_{-1}] \cup [c_{-1}, c_0]. \end{cases}$$

At the end of the **repeat**-loop, the distribution of  $r_1$  is simply the average of these two cases, excluding  $|r_0|, |r_1| > c_0$  because of the **until**-condition (and changing the normalization appropriately). We note that

$$\begin{aligned} (x + c_1 + 1)|_{-c_1}^{-c-1} + (-x - c_{-1})|_{-c_0}^{-c-1} &= (x + c_1 + 1)|_{-c_1}^{-c_0-1} + d|_{-c_0}^{-c-1}, \\ (-x + c_1 + 1)|_{c_{-1}}^{c_1} + (x - c_{-1})|_{c_{-1}}^{c_0-1} &= d|_{c_{-1}}^{c_0} + (-x + c_1 + 1)|_{c_0+1}^{c_1}, \end{aligned}$$

so that finally we find

$$R^{(1)}(x) = K^{-1} \begin{cases} -|x| + c_1 + 1, & x \in [-c_1, -c_0 - 1] \cup [c_0 + 1, c_1], \\ d, & x \in [-c_0, c_0], \end{cases}$$

where  $K = B(B + 1) + d\Delta_0$  is the normalization constant, guaranteeing that  $\sum_{y=-\infty}^{\infty} R^{(1)}(y) = 1$ . Note that exchanging the roles of  $r_0$  and  $r_1$  within the **repeat**-clause yields the same distributions after fulfillment of the **until**-condition.  $R^{(1)}(x)$  thus describes the distribution of either after the **repeat**-loop. We establish the probability of either  $r_0$  or  $r_1$  being outside  $[-c_0, c_0]$ :

$$P^* := Pr[|r| > c_0 | r \leftarrow R^{(1)}(x)] = \frac{B(B + 1)}{B(B + 1) + d\Delta_0} \quad (8)$$

2. In the second part of the algorithm, whenever  $|r_0| > c_0$  or  $|r_1| > c_0$ , these values are reassigned to  $[-c_0, c_0]$ . For simplicity, we consider only the case  $|r_0| > c_0$ . Note that if this is the case, then since  $r_1 = s - r_0$ ,  $s \in \chi(B)$  and  $|r_1| \leq c_0$ , the counterpart to  $|r_0| > c_0$  is the “flipped”

$$r_1 \in F(x) = K^{-1} \left( (-x - c_{-1})|_{-c_0}^{-c-1-1} + (x - c_{-1})|_{c_{-1}+1}^{c_0} \right).$$

We distinguish two cases, depending on the random parameter  $b$ .

- If  $b = 1$ , we simply redefine  $r_0 = -r_1$ , which amounts to  $R_1^{(2)}(x) = R_0^{(1)}(x)|_{-c_0}^{c_0} + F(x) = dK^{-1}\Theta_x[-c_0, c_0] + F(x)$ .
- If  $b = 0$ ,  $r_1$  is first resampled from  $\chi(c_0)$ , then we redefine  $r_0 = -r_1$ , which means  $F(x)$  is subtracted from  $\chi(c_0)$ , then resampled from  $\chi(c_0)$ .

In terms of the distributions, this implies  $R_1^{(2)}(x) = (1 + P^*)\chi(c_0) - F(x)$ . Averaging over both cases, we get

$$R^{(3)}(x) = \frac{1}{2} \left( R^{(2)}(x) + R'^{(2)}(x) \right) = \Delta_0^{-1}\Theta_x[-c_0, c_0],$$

which is the uniform distribution in  $[-c_0, c_0]$ . Again, this distribution holds for both  $r_0$  and  $r_1$ , which proves the lemma.

**Distribution and Expected Value of  $r^*$ .** We analyze how  $r^*$  is distributed at the end of Algorithm 1. Since  $r^*$  either takes the value of  $s$  or is zero, we first establish the probability of non-zero  $r^*$  for a given  $s$ . With the same considerations as in the proof of Lemma 1, we find that after the **repeat**-loop, the

probability of  $r_0$  or  $r_1$  being outside  $[-c_0, c_0]$  is  $P_s^* = \frac{|s|}{\Delta_0 + |s|}$ , for fixed  $s$ . Since either  $r_0$  or  $r_1$  can be outside  $[-c_0, c_0]$ ,  $r^*$  has probability  $2P_s^*$  of taking the value of  $s$  and probability  $1 - 2P_s^*$  of being zero. Averaging over all possible  $s$ , we find

$$R^*(x) = \frac{1}{d} \left( \frac{2|x|}{\Delta_0 + |x|} \Theta_x[-B, B] + \left( 4\Delta_0(H_{\Delta_0+B} - H_{\Delta_0}) - 2B + 1 \right) \Theta_x[0, 0] \right).$$

The second term represents the case  $r^* = 0$ , occurring when  $|r_0|, |r_1| \leq c_0$  and can be computed as  $\frac{\Theta_x[0,0]}{d} \sum_{y=-B}^B \left( 1 - \frac{2|y|}{\Delta_0 + |y|} \right)$ . We can now compute the expected value of  $|r^*|$  as

$$\text{Ex}_{R^*}(\delta, B) = \frac{1}{d} \sum_{y=-B}^B \frac{2|y|^2}{\Delta_0 + |y|} = \frac{1}{d} (2B(B+1) + 4\Delta_0(-B + \Delta_0(H_{\Delta_0+B} - H_{\Delta_0}))). \quad (9)$$

We analyze the asymptotic dependency of  $\text{Ex}_{R^*}(\delta, B)$  on  $\delta$ . The first term of (9) is just an offset, while the second term  $T := 4\Delta_0(-B + \Delta_0(H_{\Delta_0+B} - H_{\Delta_0}))$  strongly depends on  $\Delta_0 = 2\delta B + 1$ . In fact, using  $\Delta_0(H_{\Delta_0+B} - H_{\Delta_0}) = \sum_{y=1}^B \frac{1}{1+y\Delta_0^{-1}}$ , we find

$$\lim_{\delta \rightarrow \infty} T = \lim_{\delta \rightarrow \infty} -4 \sum_{y=1}^B \frac{y}{1 + y\Delta_0^{-1}} = -4 \sum_{y=1}^B y = -2B(B+1),$$

which is exactly the offset, thus  $\lim_{\delta \rightarrow \infty} \text{Ex}_{R^*}(\delta, B) \rightarrow 0$ . For  $\delta$  small, the behavior is dominated by the difference of the harmonic numbers.

Finally, equation (8) also allows us to express the estimated Hamming weight of the full vector  $r^*$  simply as  $2nP^*$ , since we have  $2n$  values  $r_{0,i}, r_{1,i}$ .

**Split and Lemma 2.** Correctness of **Split** is straightforward, since  $s_i^* + s_i' = s_i$  is always guaranteed. We analyze how the outputs are distributed. We again drop the indices  $i$  and indicate distributions by the corresponding capital letters. Since  $\#C^* = k$ , we have  $Pr[i \in C^*] = \frac{k}{n}$  for  $i \in \{1, \dots, n\}$ . It immediately follows that

$$\begin{aligned} S^*(x) &= \frac{k}{dn} \Theta_x[-B, B] + \frac{n-k}{n} \Theta_x[0, 0], \\ S'(x) &= \frac{n-k}{dn} \Theta_x[-B, B] + \frac{k}{n} \Theta_x[0, 0]. \end{aligned}$$

We can determine the expected value of  $s^*$  as

$$\text{Ex}_{S^*}(B, n, k) = \frac{k}{dn} B(B+1) = \frac{k}{n} \text{Ex}_S(B). \quad (10)$$

From Algorithm 2, it immediately follows that an entry  $s_i'$  is either uniform in  $[-B_i, B_i]$  or zero. Following Lemma 1, the first case results in  $r_{0,i}$  and  $r_{1,i}$  being uniform. If  $s_i' = 0$  this also immediately follows from the first **repeat**-loop in Algorithm 1, thus proving Lemma 2.