DeCSIDH: Delegating isogeny computations in the CSIDH setting

Robi Pedersen

imec-COSIC, ESAT, KU 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 20 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 in $\mathbb{Z}_{\#\mathrm{Cl}(\mathcal{O})}$) 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

* This work was supported in part by the Research Council KU Leuven grant C14/18/067, and by CyberSecurity Research Flanders with reference number VR20192203. Date of this document: May 27, 2021.

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 [25] 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, 24], 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 concurrent work. While isogeny-based protocols profit from the lowest key sizes of any of the current post-quantum standardization proposals [1, 7, 16, 27, 20], 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, 30], the delegation of post-quantum cryptographic protocols is a very new topic. In 2019, Pedersen and Uzunkol [21] proposed the first delegation algorithms for isogeny computations and improved upon their work with a follow-up paper in 2021 [22]. Their approach is applied to SIDH-type protocols, i.e. supersingular isogeny protocols over $\mathbb{F}_{p^2}$, and 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 [22] 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 *isogeny computation algorithm* CIso, which allows to delegate the computation of an isogeny, while keeping the kernel hidden from the auxiliary servers, and
   - The *hidden isogeny computation algorithm* HIso, 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 instantiations of CSIDH [4, 7, 9] and to the signature algorithms SeaSign [13] and CSI-FiSh [3] and compute the reduced cost of the delegator as compared to the local computation. Most of these 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 all the way down to 5% of the local cost. Our protocols present a communication-computation cost trade-off. Even for low 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. We support our results with benchmarks.

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

Acknowledgements. The author would like to thank Frederik Vercauteren and Osmanbey Uzunkol for helpful discussions and valuable feedback concerning this work.

2 Background

Isogeny-based cryptographic protocols are 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 [25, 28] 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 $\mathrm{Cl}(\mathcal{O})$ uniquely define subgroups via their kernel and therefore uniquely define isogenies, i.e. for a given elliptic curve $E/\mathbb{F}_p$ and ideal $\mathfrak{a} \in \mathrm{Cl}(\mathcal{O})$, we have a separable isogeny $E \to E/\mathfrak{a}$ with kernel $\bigcap_{\alpha \in \mathfrak{a}} \ker \alpha$. As a result, the ideal-class group $\mathrm{Cl}(\mathcal{O})$ 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 $E \to \mathfrak{a} * E$.

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 ℓ_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 π 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 Vélu formulae [31] and are efficient for small ℓ_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. $\mathbf{a} = (a_1, \ldots, a_n)$ 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 $-\mathbf{a}$. Note that $\mathfrak{a}_1\mathfrak{a}_2$ corresponds to $\mathbf{a}_1 + \mathbf{a}_2$.

The class group. While the class group has asymptotic size $\#\mathrm{Cl}(\mathcal{O}) \approx 2\sqrt{p}$ [26], 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 $\mathbf{a}$ from a range $\{-5, \ldots, 5\}$ 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 $\#\mathrm{Cl}(\mathcal{O}) \approx 2^{257}$. This knowledge allows to sample random elements from $\mathbb{Z}_{\#\mathrm{Cl}(\mathcal{O})}$ and transform them into vectors $\mathbf{a}$ 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 $\mathrm{Cl}(\mathcal{O})$ can then be represented by vectors $\mathbf{a} \in \mathbb{B}(N)$, where typically $N \leq \#\mathrm{Cl}(\mathcal{O})$. 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 $\mathrm{Cl}(\mathcal{O})$ 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 $\mathbb{Z}_{\#\mathrm{Cl}(\mathcal{O})}$ in the standard font (e.g. $a, b, s, \dots$). Note that we assume $\mathrm{Cl}(\mathcal{O})$ to be cyclic with publicly known generator $\mathfrak{g}$.[1] We always assume elements using the same letters to be related, e.g. $a \in \mathbb{Z}_{\#\mathrm{Cl}(\mathcal{O})}$ and $\mathbf{a} \in \mathbb{B}(N)$ will always represent $\mathfrak{a} \in \mathrm{Cl}(\mathcal{O})$, 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:

$$\mathfrak{a} = \prod_{i=1}^{n} \mathfrak{l}_i^{a_i} = \mathfrak{g}^a \,.$$

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

It is useful to note that multiplications between elements in $\mathrm{Cl}(\mathcal{O})$ naturally translate to additions in $\mathbb{Z}_{\#\mathrm{Cl}(\mathcal{O})}$ 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 $\mathfrak{a} \in Cl(\mathcal{O})$ such that $E' = \mathfrak{a} * E$.*

Classical security is based on a meet-in-the-middle attack, where the attacker generates paths of length $n/2$ from both curves and succeeds if they meet. The query complexity of this attack is $O(\sqrt{\#\mathrm{Cl}(\mathcal{O})})$. Quantum security of CSIDH is

[1] Throughout this work, we will only consider the known class group established in [3]. In any other case, where $\mathrm{Cl}(\mathcal{O})$ 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 $\mathrm{Cl}(\mathcal{O})$.

still subject to scrutiny. For current estimates of the quantum security, we refer the reader to [4, 7, 9] and [23]. 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.

Evaluation cost. Remember that for efficiency reasons, isogenies are computed using the actions of low degree isogenies defined by the ideals $\mathfrak{l}_1, \ldots, \mathfrak{l}_n$. We define I_{ℓ_i} as the cost of computing an isogeny of prime degree ℓ_i. The cost of an isogeny given by the ideal $\mathfrak{a} = \prod_{i=1}^{n} \mathfrak{l}_i^{a_i}$ can be computed as the sum of a_i consecutive ℓ_i-isogenies for $i = 1, \ldots, n$, with the order being not important. We therefore define

$$I(\mathfrak{a}) = I\left(\prod_{i=1}^{n} \mathfrak{l}_i^{a_i}\right) = \sum_{i=1}^{n} a_i I_{\ell_i}$$

as the cost of evaluating the isogeny given by $\mathfrak{a}$. Note that this notation is only heuristic and implicitly includes the operations needed in evaluating the class group action (cf. Algorithm 2 in [7]), such as point generation, point mapping and scalar multiplication, in the parameters I_{ℓ_i}. For concrete estimates of these costs, we refer the reader to e.g. [19] and to [2]; in this work, we will not need explicit estimates for I_{ℓ_i} as our results are expressed as relative costs only.

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 $\mathcal{U}$. However 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 adversarily produced outputs could only fulfill with a low probability.

Definition 2 (Outsource-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* outsource-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].

Two important parameters to assess delegations are the reduction in computational cost α that $\mathcal{T}$ profits from, when compared to the local computation of Alg, as well as the degree of certainty β that the outputs of the servers are correct. These are formalized in the following definition.

Definition 3 ((α, β)-outsource-security [15]). *A pair of algorithms $(T,\mathcal{U})$ are an (α, β)-outsource secure implementation of an algorithm* Alg*, if*

- *$(\mathcal{T},\mathcal{U})$ are an outsource-secure implementation of* Alg*,*
- *for all inputs x, the running time of $\mathcal{T}$ is at most an α-multiplicative factor of the running time of* Alg(x) *(i.e. Time$(\mathcal{T}) \leq \alpha$Time(*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 $\geq \beta$.*

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 following assumptions.

Definition 4 (1HBC [10]). *The* one honest-but-curious program model *defines the adversary as $\mathcal{A} = (\mathcal{E},\mathcal{U})$, where $\mathcal{U}$ consists of a single server that always returns correct results, but may try to extract sensitive data.*

Definition 5 (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.*

We further define the following model, based on Definition 4:

Definition 6 (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 $(\mathfrak{a}_1, E_1), \ldots, (\mathfrak{a}_k, E_k)$ consisting of ideals $\mathfrak{a}_i$ and associated elliptic curves E_i. The servers then generate and return the codomain curves $\mathfrak{a}_i * E_i$ for each $i = 1, \ldots, k$. We write

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

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

4 Shrouding

4.1 The ShrVec algorithm

Before we present implementations for our delegation algorithms, we discuss how to shroud ideals. The basic idea is to split the secret $\mathfrak{s}$ into a pair of randomly looking ideals $(\mathfrak{a}_1, \mathfrak{a}_2)$, so that $\mathfrak{a}_1 * (\mathfrak{a}_2 * E) = \mathfrak{s} * E$. In the case where $\mathrm{Cl}(\mathcal{O})$ is known, we can simply generate $(a, s - a)$ for $a \leftarrow \mathbb{Z}_{\#\mathrm{Cl}(\mathcal{O})}$.

On the other hand, if $\mathrm{Cl}(\mathcal{O})$ is unknown, we cannot simply generate $(\mathbf{a}, \mathbf{s} - \mathbf{a})$ for a random vector $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{B}$ since $\mathbf{s} - \mathbf{a}$ would no longer be in $\mathbb{B}$ and leak information about the secret [29]. 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 $i \in \{1, \ldots, n\}$ 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 by using the procedure defined in Algorithm 1. Let $\chi(k)$ denote the uniform distribution in $[-k, k]$ and let $\delta\mathbb{B}(N) = [-\delta_1 B_1, \delta_1 B_1] \times \cdots \times [-\delta_n B_n, \delta_n B_n]$.

Notation. We write the invocation of Algorithm 1 as $(\mathbf{r_0}, \mathbf{r_1}, \mathbf{r}^*) \leftarrow \mathsf{ShrVec}_\delta(s)$. We generally omit δ in the index if it is clear from the context or not explicitly needed.

Input : secret $\mathbf{s} = (s_1, \ldots, s_n) \in \mathbb{B}$ and parameters $\delta = (\delta_1, \ldots, \delta_n)$
Output: $\mathbf{r_0}, \mathbf{r_1} \in \delta\mathbb{B}$ uniform, $\mathbf{r}^* \in \mathbb{B}$ small, such that $\mathbf{r_0} + \mathbf{r_1} + \mathbf{r}^* = \mathbf{s}$

for $i = 1, \ldots, n$ **do**
  repeat
    $r_{0,i} \leftarrow \chi((\delta_i + 1)B_i)$
    $r_{1,i} = s_i - r_{0,i}$
  until $|r_{0,i}| \leq \delta_i B_i$ *or* $|r_{1,i}| \leq \delta_i B_i$

  $b \leftarrow \{0, 1\}$
  if $|r_{0,i}| > \delta_i B_i$ **then**
    if $b == 0$ **then** $r_{1,i} \leftarrow \chi(\delta_i B_i)$
    $r_{0,i} = -r_{1,i}$
    $r_i^* = s_i$
  else if $|r_{1,i}| > \delta_i B_i$ **then**
    if $b == 0$ **then** $r_{0,i} \leftarrow \chi(\delta_i B_i)$
    $r_{1,i} = -r_{0,i}$
    $r_i^* = s_i$
  else $r_i^* = 0$
end
return $\mathbf{r_0} = (r_{0,1}, \ldots, r_{0,n})$, $\mathbf{r_1} = (r_{1,1}, \ldots, r_{1,n}), \mathbf{r}^* = (r_1^*, \ldots, r_n^*)$.

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

Analysis. We analyze different properties of Algorithm 1. The following holds for any $i \in \{1, \ldots, n\}$, we will therefore omit the index i. Let $\Theta[x]$ denote the (right-continuous) discrete Heaviside step function. We define the rectangular function $\Theta_x[x_{\text{start}}, x_{\text{end}}] := \Theta[x - x_{\text{start}}] - \Theta[x - x_{\text{end}} - 1]$ for $x_{\text{start}} \leq x_{\text{end}}$. We also write $f(x)\big|_{x_{\text{start}}}^{x_{\text{end}}} = f(x)\Theta_x[x_{\text{start}}, x_{\text{end}}]$ as a shorthand. For further conciseness in notation, we introduce

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

as well as $d = 2B + 1$. 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

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

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 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

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

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

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

Lemma 1. *Algorithm 1 is correct.*

Proof. 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. □

Lemma 2. *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* $\delta\mathbb{B}(N)$*.*

Proof. In order to prove this lemma, 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 $^{(i)}$.

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

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

   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

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

so that finally we find

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

where $K = B(B+1) + d\Delta_0$ is the normalization constant, guaranteeing that $\sum_{y=-\infty}^{\infty} R^{(1)}(y) = 1$. We 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. Thus, $R^{(1)}(x)$ describes the distribution of either r_0 and r_1 after the `repeat`-loop. We establish the probability of either r_0 or r_1 being outside $[-c_0, c_0]$:

$$P^* := Pr\Big[|r| > c_0 \Big| r \leftarrow R^{(1)}(x)\Big] = \frac{B(B+1)}{B(B+1) + d\Delta_0} \tag{2}$$

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"

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

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

- If $b = 1$, we simply redefine $r_0 = -r_1$, which amounts to $R_1^{(2)}(x) = R_0^{(1)}(x)\big|_{-c_0}^{c_0} + F(x) = dK^{-1}\Theta_x[-c_0, c_0] + F(x)$.
- 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.

□

4.2 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 2, 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. Note that by substituting $|s|$ with $\mathsf{Ex}_S(B)$, we immediately recover equation (2). 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

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

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

$$\mathsf{Ex}_{R^*}(\delta, B) = \frac{1}{d}\sum_{y=-B}^{B}\frac{2|y|^2}{\Delta_0 + |y|} = \frac{1}{d}\big(2B(B+1) + 4\Delta_0(-B + \Delta_0(H_{\Delta_0+B} - H_{\Delta_0}))\big). \tag{3}$$

We analyze the asymptotic dependency of $\mathsf{Ex}_{R^*}(\delta, B)$ on δ. The first term of (3) is just an offset, while the second term strongly depends on $\Delta_0 = 2\delta B + 1$. In fact,

$$\Delta_0(H_{\Delta_0+B} - H_{\Delta_0}) = \sum_{y=1}^{B}\frac{1}{1 + y\Delta_0^{-1}} \quad < \quad \sum_{y=1}^{B} 1 = B,$$

since $y\Delta_0^{-1} > 0$, which means that the second term in (3) is negative. Using

$$4\Delta_0\big(-B + \Delta_0(H_{\Delta_0+B} - H_{\Delta_0})\big) = -4\sum_{y=1}^{B}\frac{y}{1 + y\Delta_0^{-1}},$$

we can compute its limit for large δ:

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

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

4.3 Splitting s

We present Algorithm 2, which splits a vector $\mathbf{s}$ into two vectors $\mathbf{s}'$ and $\mathbf{s}^*$, so that $\mathbf{s}^*$ has a given Hamming weight.

Correctness of the algorithm is straightforward. 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

Input : secret $\mathbf{s} = (s_1, \ldots, s_n) \in \mathbb{B}$, and parameter k
Output: $\mathbf{s}', \mathbf{s}^* \in \mathbb{B}$, such that $\mathbf{s}^*$ has Hamming weight $\leq k$ and $\mathbf{s}' + \mathbf{s}^* = \mathbf{s}$.

Sample a uniform subset $C^* \leftarrow \{1, \ldots, n\}$ of size k.
for $i = 1, \ldots, n$ **do**
  if $i \in C^*$ **then**
    $(s_i^*, s_i') = (s_i, 0)$
  else
    $(s_i^*, s_i') = (0, s_i)$
  end
end
return $\mathbf{s}^* = (s_1^*, \ldots, s_n^*), \mathbf{s}' = (s_1', \ldots, s_n')$.

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

$i \in \{1, \ldots, n\}$. It immediately follows that

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

We can determine the expected value of s^* as

$$\mathsf{Ex}_{S^*}(B, n, k) = \frac{k}{dn}B(B+1) = \frac{k}{n}\mathsf{Ex}_S(B)\,. \tag{4}$$

Lemma 3. *Let* $(\mathbf{s}^*, \mathbf{s}') \leftarrow \mathsf{Split}(\mathbf{s}, k)$ *where* $\mathbf{s} \leftarrow \mathbb{B}(N)$ *uniform and let* $(\mathbf{r}_0, \mathbf{r}_1, \mathbf{r}^*) \leftarrow \mathsf{ShrVec}_\delta(\mathbf{s}')$. *Then, the outputs* $\mathbf{r_0}$ *and* $\mathbf{r_1}$ *of Algorithm 1 are uniformly distributed in* $\delta\mathbb{B}(N)$.

Proof. From Algorithm 2, it immediately follows that an entry s_i' is either uniform in $[-B_i, B_i]$ or zero. Following Lemma 2, 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. □

4.4 Cost reduction functions

Using the expectation values established in equations (1), (3) and (4), we define the following functions:

$$\alpha(\delta, B, n, k, b) := \frac{(1+b)\mathsf{Ex}_{R^*}(\delta, B) + \mathsf{Ex}_{S^*}(B, n, k)}{\mathsf{Ex}_S(B)}\,, \quad \alpha_{\mathcal{U}}(\delta, B) := \frac{\mathsf{Ex}_R(\delta, B)}{\mathsf{Ex}_S(B)}\,, \tag{5}$$

which we call the *cost reduction functions*. Later in Section 5, the parameter $b \in \{0, 1\}$ will allow us to distinguish between the 2HBC and OMTUP cases. Assuming B, n and k fixed, these functions asymptotically behave as

$$\alpha(\delta, B, n, k, b) = O\left(b\delta^2(H_{\Delta_0+B} - H_{\Delta_0})\right) \quad \text{and} \quad \alpha_{\mathcal{U}}(\delta, B) = O(\delta)\,.$$

Note also that $\lim_{\delta\to\infty}\alpha(\delta,B,n,k,b) = \frac{\mathsf{Ex}_{S^*}(B,n,k)}{\mathsf{Ex}_S(B)}$. Cost reduction functions will later allow us to estimate the relative cost of isogenies. For instance, let $\mathfrak{s}=\prod_{i=1}^n \mathfrak{l}_i^{s_i}$ and $\mathfrak{r}=\prod_{i=1}^n \mathfrak{l}_i^{r_i}$, and let $a_i = r_i/s_i$ for all $i\in\{1,\dots,n\}$. Then, we can express the relative cost of $I(\mathfrak{r})$ and $I(\mathfrak{s})$ as follows:

$$\frac{I(\mathfrak{r})}{I(\mathfrak{s})} = \frac{\sum_{i=1}^n r_i I_{\ell_i}}{\sum_{i=1}^n s_i I_{\ell_i}} = \frac{\sum_{i=1}^n a_i s_i I_{\ell_i}}{\sum_{i=1}^n s_i I_{\ell_i}}\,,$$

If r_i and s_i are not fixed, but rather sampled randomly, we can still express this ratio using the expected values $\mathsf{Ex}_R(\delta_i,B_i)$ and $\mathsf{Ex}_S(B_i)$, respectively. We then find

$$\frac{I(\mathfrak{r})}{I(\mathfrak{s})} \approx \frac{\sum_{i=1}^n \mathsf{Ex}_R(\delta_i,B_i) I_{\ell_i}}{\sum_{i=1}^n \mathsf{Ex}_S(B_i) I_{\ell_i}} = \frac{\sum_{i=1}^n \alpha_{\mathcal{U}}(\delta_i,B_i)\mathsf{Ex}_S(B_i) I_{\ell_i}}{\sum_{i=1}^n \mathsf{Ex}_S(B_i) I_{\ell_i}}\,.$$

We are especially interested in the case where all $\delta_1 = \cdots = \delta_n$ and all $B_1 = \cdots = B_n$. In that case, we can express

$$\alpha(\delta,B,n,k,b) = \frac{(1+b)I(\mathfrak{r}^*) + I(\mathfrak{s}^*)}{I(\mathfrak{s})} \quad \text{and} \quad \alpha_{\mathcal{U}}(\delta,B) = \frac{I(\mathfrak{r})}{I(\mathfrak{s})}\,.$$

Thus, for equal δ_i and B_i, $\alpha_{\mathcal{U}}(\delta,B)$ represents the relative cost of the action of an ideal $\mathfrak{r}$, represented by the vector $\mathbf{r}$, to the action of $\mathfrak{s}$ on an elliptic curve, while $\alpha(\delta,B,n,k,b)$ expresses the relative cost of computing the action $\mathfrak{s}^*$ and $(1+b)$ times the action $\mathfrak{r}^*$ versus the action of $\mathfrak{s}$.

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 $\mathbf{s}=(s_1,\dots,s_n)\in\mathbb{B}(N)$ of $\mathfrak{s}=\prod_{i=1}^n \mathfrak{l}^{s_i}$. In the case where $\mathrm{Cl}(\mathcal{O})$ is known, we further assume that $s\in\mathbb{Z}_{\#\mathrm{Cl}(\mathcal{O})}$ is known by the delegator, such that $\mathfrak{s}=\mathfrak{g}^s$. We define the two algorithms below, using the formalism from [15].

Definition 7 (CIso and HIso). *The* isogeny computation algorithm CIso *and the* hidden isogeny computation algorithm HIso *take as inputs a supersingular elliptic curve E defined over $\mathbb{F}_p$ and an ideal $\mathfrak{s}$, either as an element in $\mathbb{Z}_{\#Cl(\mathcal{O})}$ 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*

$$\mathfrak{s}*E \leftarrow \mathsf{CIso}(\mathfrak{s},E) \quad \textit{and} \quad \mathfrak{s}*E \leftarrow \mathsf{HIso}(\mathfrak{s},E)\,.$$

Below, we motivate and present implementations for both CIso and HIso in the OMTUP and 2HBC assumptions (Definitions 5 and 6). All algorithms work in two rounds of delegation.

5.1 CIso: Unprotected codomain

The main concept of delegating with CIso is to keep $\mathfrak{s}$ hidden from the servers. Our general approach to shroud $\mathfrak{s}$ 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. But in the case where 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 $\mathfrak{x}$ to its result in both rounds yielding the result $\mathfrak{x} * (\mathfrak{s} * E)$ 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 tuples $(\mathfrak{c}_1, \mathfrak{c}_2, \mathfrak{d}_1, \mathfrak{d}_2)$ of ideals that satisfy $\mathfrak{c}_1\mathfrak{c}_2 = \mathfrak{d}_1\mathfrak{d}_2$. In the case where we work over $\mathbb{Z}_{\#\mathrm{Cl}(\mathcal{O})}$, this is straightforward. If we work with elements in $\delta\mathbb{B}(N)$, we can implement this as follows: for $i = 1, \ldots, n$, 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.

CIso: 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 $\mathrm{Cl}(\mathcal{O})$ is known, generate $a_1 \leftarrow \mathbb{Z}_{\#\mathrm{Cl}(\mathcal{O})}$ uniformly at random and compute $a_2 = s - a_1$.
   (b) If $\mathrm{Cl}(\mathcal{O})$ is not known, generate $(\mathbf{a_1}, \mathbf{a_2}, \mathbf{a}^*) \leftarrow \mathsf{ShrVec}(s)$.
2. Delegate the computation of $E_1 \leftarrow \mathcal{U}_1((\mathfrak{a}_1, E))$.
3. In the case where $\mathrm{Cl}(\mathcal{O})$ 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 CIso in the 2HBC assumption.

5.2 HIso: Hidden codomain

Next to keeping $\mathfrak{s}$ hidden, HIso also does not disclose the codomain curve to the auxiliary servers. The idea works similar to CIso, 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 $\mathrm{Cl}(\mathcal{O})$ is known, this is achieved by working modulo the relation lattice [3].[2] We then call $\widetilde{\mathbb{B}}(N) \subseteq \mathbb{B}(N)$ the subset from which $\mathbf{s}^*$ is sampled. We can achieve this splitting of $\mathbf{s}$ using the Split-procedure (Algorithm 2). The protocol then uses CIso as a subroutine with $\mathbf{s}'$ 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 CIso is called with the appropriate assumption.

[2] Note that $\mathbb{B}(N)$ does not necessarily contain a representation for all elements in $\mathrm{Cl}(\mathcal{O})$. We ignore this case here and assume we can still 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 $\mathrm{Cl}(\mathcal{O})$ is known, generate two random elements $a_1, b_1 \leftarrow \mathbb{Z}_{\#\mathrm{Cl}(\mathcal{O})}$ and compute $a_2 = s - a_1$ and $b_2 = s - b_1$.
   (b) If $\mathrm{Cl}(\mathcal{O})$ is not known, generate $(\mathbf{a_1}, \mathbf{a_2}, \mathbf{a}^*) \leftarrow \mathsf{ShrVec}(s)$ and $(\mathbf{b_1}, \mathbf{b_2}, \mathbf{b}^*) \leftarrow \mathsf{ShrVec}(s)$.
   Further, generate a set of random ideals $\mathcal{R} = \{\mathfrak{e} \mid \mathfrak{e} \leftarrow \mathrm{Cl}(\mathcal{O})\}$ and a set of random ideal tuples $\mathcal{S} = \{(\mathfrak{c}_1, \mathfrak{c}_2, \mathfrak{d}_1, \mathfrak{d}_2) \leftarrow \mathrm{Cl}(\mathcal{O})^4 \mid \mathfrak{c}_1\mathfrak{c}_2 = \mathfrak{d}_1\mathfrak{d}_2\}$, where all the ideals are generated using $\mathbb{Z}_{\#\mathrm{Cl}(\mathcal{O})}$ or $\delta\mathbb{B}(N)$, respectively.
2. Delegate the computation of

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

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

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

5. Delegate the computation of

$$E_s, \{E_{\mathfrak{d}}\} \leftarrow \mathcal{U}_1\Big((\mathfrak{b}_2, E_{\mathfrak{b}_1}), \{(\mathfrak{d}_2, E_{\mathfrak{d}_1}) \mid \mathfrak{d}_2 \in \mathcal{S}\}\Big),$$
$$E'_s, \{E_{\mathfrak{c}}\} \leftarrow \mathcal{U}_2\Big((\mathfrak{a}_2, E_{\mathfrak{a}_1}), \{(\mathfrak{c}_2, E_{\mathfrak{c}_1}) \mid \mathfrak{c}_2 \in \mathcal{S}\}\Big).$$

6. Verify if $E_s \stackrel{?}{=} E'_s$ 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.

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 $\widetilde{\mathbb{B}}(N) \subseteq \mathbb{B}(N)$ of size $\widetilde{D} = \#\widetilde{\mathbb{B}}(N)$, that is big enough that searching the entire space is at least as hard as breaking a GAIP instance. Since the servers are only given $\mathfrak{s}' * E$, they can not 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 $\widetilde{D}$ 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 $O\big(\widetilde{D}^{1/2}\big)$. Thus in order to ensure a quantum security level of λ,

HIso: **General case**

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

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

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

we choose $\widetilde{D}^{1/2} = 2^\lambda$, which corresponds to $\widetilde{D} = 2^{2\lambda}$. We can therefore define $\widetilde{\mathbb{B}}(N)$ analogously to $\mathbb{B}(N)$, i.e.

$$\widetilde{\mathbb{B}}(N) = \widetilde{\mathbb{B}}_1 \times \cdots \times \widetilde{\mathbb{B}}_n \,,$$

where $\widetilde{\mathbb{B}}_i \in \{[0,0], \mathbb{B}_i\}$ of size $\widetilde{d}_i \in \{1, d_i\}$, such that $\widetilde{D} = \prod_{i=1}^n \widetilde{d}_i \approx 2^{2\lambda}$.

The input parameter k of Split determines the number of non-zero $\widetilde{\mathbb{B}}_i$. 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 3, the adversary can not distinguish in which entries $\mathbf{s}'$ 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

$$\binom{n}{k} \prod_{i \in C^*} d_i \approx 2^{2\lambda} \,. \tag{6}$$

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. $E_s \neq s * E$. 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 $\kappa \in \{1, \ldots, 1+m_s\}$ 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 $\binom{m_s}{\kappa-1} \Big/ \binom{1+m_s+m_r}{\kappa}$. Furthermore, in the second round, the malicious server has to identify the *same* subset, which it achieves with probability $1 \Big/ \binom{1+m_s}{\kappa}$, yielding the full success

probability for the adversary of

$$Pr[success] = \frac{\binom{m_s}{\kappa-1}}{\binom{1+m_s+m_r}{\kappa}\binom{1+m_s}{\kappa}} = \frac{\kappa}{1+m_s}\frac{\kappa!(m_s+m_r+1-\kappa)!}{(m_s+m_r+1)!}\,. \tag{7}$$

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

$$Pr[success \mid m_r \geq 3] = \frac{1}{(1+m_s)(1+m_s+m_r)}\,.$$

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

$$\beta(m_s) = 1 - Pr[success \mid m_r = 3] = \frac{m_s^2 + 5m_s + 3}{m_s^2 + 5m_s + 4}\,. \tag{8}$$

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 $\mathfrak{u}_1, \mathfrak{u}_2$ either as elements in $\mathbb{Z}_{\#\mathrm{Cl}(\mathcal{O})}$ (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 $E_1 \leftarrow \mathcal{U}_1((\mathfrak{u}_1, E))$, computes $E_1 \leftarrow \mathbf{u}^* * E_1$ if applicable, then makes the query $E_2 \leftarrow \mathcal{U}_2((\mathfrak{u}_2, E_1))$. $\mathcal{S}_1$ returns E_2 and saves its own state and those of the servers. In any round, the input values $\mathfrak{u}_1, \mathfrak{u}_2$ are indistinguishable from $\mathfrak{a}_1, \mathfrak{a}_2$. In case (b), this is given by Lemma 2.
- 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 $\mathfrak{s}$. $\mathcal{UVIEW}_{\mathrm{real}} \sim \mathcal{UVIEW}_{\mathrm{ideal}}$ is guaranteed by the indistinguishability of $\mathfrak{u}_1, \mathfrak{u}_2, \mathfrak{u}^*$ and $\mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{a}^*$. Note that applying $\mathfrak{a}^* * E_1$ 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 $\mathfrak{a}^*$. □

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 $E_{\mathfrak{c}} \stackrel{?}{=} E_{\mathfrak{d}}$ 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 $\mathfrak{u}_1, \mathfrak{u}_2, \mathfrak{v}_1, \mathfrak{v}_2$ and in case (b) further $\mathfrak{u}^*, \mathfrak{v}^*$ 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 $(\mathfrak{n}_1, \mathfrak{n}_2, \mathfrak{n}_3, \mathfrak{n}_4)_i \in \mathcal{N}_1 \times \mathcal{N}_2 \times \mathcal{N}_3 \times \mathcal{N}_4$, it holds that $\mathfrak{n}_1\mathfrak{n}_4 = \mathfrak{n}_2\mathfrak{n}_3$, pairwise for $i = 1, \ldots, m_s$. Then $\mathcal{S}_1$ makes the queries

$$E_{\mathfrak{u}_1}, \{E_{\mathfrak{n}_1}\}, \{E_{\mathfrak{m}_1}\} \leftarrow \mathcal{U}_1\Big((\mathfrak{u}_1, E), \{(\mathfrak{n}_1, E) \mid \mathfrak{n}_1 \in \mathcal{N}_1\}, \{(\mathfrak{m}_1, E) \mid \mathfrak{m}_1 \in \mathcal{M}_1\}\Big),$$
$$E_{\mathfrak{v}_1}, \{E_{\mathfrak{n}_2}\}, \{E_{\mathfrak{m}_2}\} \leftarrow \mathcal{U}_2\Big((\mathfrak{v}_1, E), \{(\mathfrak{n}_2, E) \mid \mathfrak{n}_2 \in \mathcal{N}_2\}, \{(\mathfrak{m}_2, E) \mid \mathfrak{m}_2 \in \mathcal{M}_2\}\Big).$$

$\mathcal{S}_1$ verifies the results. If either of the elements in $\{E_{\mathfrak{m}_1}\}$ or $\{E_{\mathfrak{m}_2}\}$ are incorrect, then $\mathcal{S}_1$ returns $\perp$, otherwise it continues. In case (b), $\mathcal{S}_1$ computes $E_{\mathfrak{u}_1} \leftarrow \mathfrak{u}^* * E_{\mathfrak{u}_1}$ and $E_{\mathfrak{v}_1} \leftarrow \mathfrak{v}^* * E_{\mathfrak{v}_1}$. Then, in the second round, $\mathcal{S}_1$ makes the queries

$$E_{\mathfrak{v}_2}, \{E_{\mathfrak{n}_3}\}, \leftarrow \mathcal{U}_1\Big((\mathfrak{v}_2, E_{\mathfrak{v}_1}), \{(\mathfrak{n}_3, E_{\mathfrak{n}_2}) \mid \mathfrak{n}_3 \in \mathcal{N}_3\}\Big),$$
$$E_{\mathfrak{u}_2}, \{E_{\mathfrak{n}_4}\}, \leftarrow \mathcal{U}_2\Big((\mathfrak{u}_2, E_{\mathfrak{u}_1}), \{(\mathfrak{n}_4, E_{\mathfrak{n}_1}) \mid \mathfrak{n}_4 \in \mathcal{N}_4\}\Big).$$

Again, $\mathcal{S}_1$ verifies the results. If $\nexists \mathfrak{x} : E_{\mathfrak{u}_2} = (\mathfrak{x}\mathfrak{u}_1\mathfrak{u}_2) * E \wedge E_{\mathfrak{v}_2} = (\mathfrak{x}\mathfrak{v}_1\mathfrak{v}_2) * E$, $\mathcal{S}_1$ returns $\perp$. Otherwise, let κ be the number of pairs $(E_{\mathfrak{n}_3}, E_{\mathfrak{n}_4})$ for which there doesn't exist such an $\mathfrak{x}$. Then with probability $1 - Pr[success]$ (as given in equation (7)), $\mathcal{S}_1$ 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 $(\mathfrak{u}_1, \mathfrak{u}_2, \mathfrak{u}^*, \mathfrak{v}_1, \mathfrak{v}_2, \mathfrak{v}^*, \mathcal{M}_1, \mathcal{M}_2, \mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3, \mathcal{N}_4)$ is indistinguishable from the tuple $(\mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{a}^*, \mathfrak{b}_1, \mathfrak{b}_2, \mathfrak{b}^*, \mathcal{R}, \mathcal{R}, \{\mathfrak{c}_1 \in \mathcal{S}\}, \{\mathfrak{d}_1 \in \mathcal{S}\}, \{\mathfrak{d}_2 \in \mathcal{S}\}, \{\mathfrak{c}_2 \in \mathcal{S}\})$, due to uniform sampling or because of Lemma 2. 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 $\mathcal{EVIEW}_{\text{real}} \sim \mathcal{EVIEW}_{\text{ideal}}$.
– 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. $\mathcal{UVIEW}_{\text{real}} \sim \mathcal{UVIEW}_{\text{ideal}}$ is guaranteed by the indistinguishability of the tuple described above. □

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 (6). □

Remark 1. Note that Definition 6 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 $\mathfrak{a}_1$.

Definition 5 implies that at least one of the two servers is honest, so that collusion is not possible in the OMTUP case.

Computational costs. We establish the computational cost for the delegator for CIso and HIso in the 2HBC and OMTUP assumptions. We define the bit b as an indicator to distinguish between the 2HBC ($b = 0$) and the OMTUP ($b = 1$) assumptions.

After the first delegation round of CIso, if $\mathrm{Cl}(\mathcal{O})$ is unknown, the delegator has to compute $1 + b$ isogenies of size $I(\mathfrak{r}^*) = \sum_{i=1}^{n} \mathsf{Ex}_{R^*}(\delta_i, B_i) I_{\ell_i}$, given by equation (3). In the case of HIso, the delegator further has to compute an isogeny of size $I(\mathfrak{s}^*) = \sum_{i=1}^{n} \mathsf{Ex}_{S^*}(B_i, n, k) I_{\ell_i}$, given by equation (4). We ignore the costs of comparison operations, point generation in $\mathbb{Z}_{\#\mathrm{Cl}(\mathcal{O})}$ and invoking ShrVec, as they are negligible in comparison to isogeny computations. We find the relative cost of the delegator's computation compared to the local computation:

$$\frac{I(\mathfrak{s}^*) + (1+b) I(\mathfrak{r}^*)}{I(\mathfrak{s})} = \frac{\sum_{i=1}^{n} (\mathsf{Ex}_{S^*}(B_i, n, k) + (1+b)\mathsf{Ex}_{R^*}(\delta_i, B_i)) I_{\ell_i}}{\sum_{i=1}^{n} \mathsf{Ex}_S(B_i) I_{\ell_i}} .$$

Note that in the case of CIso, we have $k = 0$ and $\mathsf{Ex}_{S^*}(B_i, n, 0) = 0$. If all $\{\delta_i\}_{i=1,\ldots,n}$ and $\{B_i\}_{i=1,\ldots,n}$ are equal, we find

$$\frac{I(\mathfrak{s}^*) + (1+b) I(\mathfrak{r}^*)}{I(\mathfrak{s})} = \frac{\mathsf{Ex}_{S^*}(B, n, k) + (1+b)\mathsf{Ex}_{R^*}(\delta, B)}{\mathsf{Ex}_S(B)} = \alpha(\delta, B, n, k, b) .$$

Each server has to compute $(2m_s + m_r + 1)b + 1$ isogenies of cost given by $I(\mathfrak{r}) = \sum_{i=1}^{n} \mathsf{Ex}_R(\delta_i, B_i) I_{\ell_i}$. In the case of equal δ_i and equal B_i we find the relative cost

$$\begin{aligned} \frac{((2m_s + m_r + 1)b + 1) I(\mathfrak{r})}{I(\mathfrak{s})} &= \frac{((2m_s + m_r + 1)b + 1)\mathsf{Ex}_R(\delta, B)}{\mathsf{Ex}_S(B)} \\ &= ((2m_s + m_r + 1)b + 1)\alpha_{\mathcal{U}}(\delta, B). \end{aligned}$$

In the case where $\mathrm{Cl}(\mathcal{O})$ is known, the $\mathfrak{r}^*$-isogeny does not need to be computed, so that we can set $\mathsf{Ex}_{R^*} = 0$. As established in Section 4.4, this is the same as considering the limit $\delta \to \infty$, so that we can define

$$\alpha_{\mathrm{Cl}(\mathcal{O})}(B, n, k) := \lim_{\delta \to \infty} \alpha(\delta, B, n, k, b) = \frac{I(\mathfrak{s}^*)}{I(\mathfrak{s})} = \frac{\mathsf{Ex}_{S^*}(B, n, k)}{\mathsf{Ex}_S(B)} .$$

Communication cost. We want to be able 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
$\mathbb{Z}_{\#\mathrm{Cl}(\mathcal{O})}$	$\lceil \log_2 \#\mathrm{Cl}(\mathcal{O}) \rceil$
$\mathbb{B}(N)$	$\sum_{i=1}^{n} \log_2 d_i$
$\delta\mathbb{B}(N)$	$\sum_{i=1}^{n} \log_2 \Delta_{0,i}$
$\mathbb{F}_p$	$\lceil \log_2 p \rceil$

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. We can estimate the communication costs by establishing the minimal number of bits of an element uniformly sampled from $\delta\mathbb{B}$ as

$$\mathsf{Ex}_I(\delta\mathbb{B}) := \sum_{i=1}^{n} \frac{1}{\Delta_{0,i}} \sum_{y=-\delta_i B_i}^{\delta_i B_i} \lceil \log_2(2|y|+1) \rceil \,.$$

Using this representation considerably lowers the communication cost, especially for large δ_i.

We can now establish the communication cost for the delegation of CIso (note that HIso has the same cost). In the 2HBC case, the delegator has to upload one element from either $\mathbb{Z}_{\#\mathrm{Cl}(\mathcal{O})}$ or $\delta\mathbb{B}(N)$ and download 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 $\mathbb{Z}_{\#\mathrm{Cl}(\mathcal{O})}$ or $\delta\mathbb{B}(N)$ to each server and downloads the same amount of elliptic curves. We define

$$\begin{aligned} Up(x) &= \begin{cases} (1+(2m_s+m_r+1)b)\lceil \log_2 \#\mathrm{Cl}(\mathcal{O}) \rceil, & x = \mathrm{Cl}(\mathcal{O}), \\ (1+(2m_s+m_r+1)b)\mathsf{Ex}_I(\delta\mathbb{B}), & x = \delta\mathbb{B}, \end{cases} \\ Down &= (1+(2m_s+m_r+1)b)\lceil \log_2 p \rceil, \end{aligned} \tag{9}$$

as the upload and download costs per server in the 2HBC ($b = 0$) and OMTUP ($b = 1$) case.

6 Applications

In this section we discuss how to apply our delegation algorithm to some of the isogeny-based protocols in the CSIDH setting.

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 $\mathbf{s} \in \mathbb{B}(N)$ 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 $E_A \leftarrow \mathsf{CIso}(\mathfrak{s}, E)$.
– *Key exchange*: Delegate $\mathfrak{s} * E_B \leftarrow \mathsf{HIso}(\mathfrak{s}, E_B)$.

Because of the simple structure of $\mathbb{B}(N)$, we can express the reduced cost for the delegator using the cost reduction functions defined in equation (5), i.e. the cost in the key generation step is given by $(1+b)I(\mathfrak{r}^*) = \alpha(\delta, B, n, 0, b)I(\mathfrak{s})$, while the cost in the key exchange step is given by $I(\mathfrak{s}^*)+(1+b)I(\mathfrak{r}^*) = \alpha(\delta, B, n, k, b)I(\mathfrak{s})$. In order to estimate the cost reduction function for the complete protocol, we define

$$\alpha_{\mathrm{CSIDH}}(\delta, B, n, k, b) = \frac{\mathsf{Ex}_{S^*}(B, n, k) + 2(1+b)\mathsf{Ex}_{R^*}(\delta, B)}{2\mathsf{Ex}_S(B)}\,, \tag{10}$$

assuming we use the same δ in both rounds.

We now turn our attention to specific instantiations of CSIDH and try to estimate the reduced cost for the delegator. While the security of CSIDH 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 λ are mainly based on the results in [4, Table 8], [9, Table 3] and [23, 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 (4) is fulfilled. Looking at the different security assessments found in the literature, we take the lower estimate of $\lambda \approx 58$ from [23, Figure 1], which corresponds to $k = 18$.

Table 1 shows the theoretically estimated cost reduction (left table) using equation (10), for the 2HBC ($b = 0$) and OMTUP ($b = 1$) assumptions. We also used the *Velusqrt* implementation of CSIDH-512,[3] introduced in [2], to benchmark the different delegation subprocesses in MAGMA (right table). Note that the case for $\delta \to \infty$ also corresponds to the cost of delegating CSIDH if the class group structure and relation lattice are known. For the CSIDH-512 parameter set this is indeed the case as the class group has been computed in [3].

[3] https://velusqrt.isogeny.org/software.html

α_{CSIDH}	$\delta=1$	5	10	100	$\to\infty$
2HBC	0.610	0.255	0.191	0.129	0.117
OMTUP	1.098	0.388	0.261	0.136	0.117

$\alpha_{\mathrm{CSIDH}}^{\mathrm{benchmark}}$	$\delta=1$	5	10	100	1000
2HBC	0.485	0.261	0.214	0.142	0.111
OMTUP	0.929	0.369	0.305	0.153	0.136

Table 1. In the left table are theoretical estimates for the cost reduction function $\alpha_{\mathrm{CSIDH}}(\delta, 5, 74, 18, b)$ from equation (10) for different δ in the 2HBC and OMTUP assumptions. The right table represents benchmarks using the *Velusqrt* software package from [2], where we define $\alpha_{\mathrm{CSIDH}}^{\mathrm{benchmark}}$ as the ratio between the number of CPU cycles the delegator has to perform during the delegated protocol execution and the number of CPU cycles in the local computation. The benchmarks were done in Magma v2.25-6 on an Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz with 128 GB memory.

We can see that the theoretical predictions in Table 1 match quite accurately with the benchmarks. Except in the case $\delta = 1$, the delegator cost is slightly higher than expected, which is due to the fact that some of the overhead of the CSIDH protocol is done by the delegator rather than the servers. The benchmarks also further 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.

Server cost. Table 2 shows the theoretical and benchmarked values for the computational costs of the auxiliary servers. The theoretical values are estimated using

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

using $\alpha_{\mathcal{U}}(\delta, B)$ from equation (5). Our predictions in Table 2 match with the benchmarks for low δ. For higher δ, the overhead of the local computation (which is taken over by the delegator) becomes more important, so that the actual relative cost of the servers is actually lower than the expected one.

$\alpha_{\mathcal{U},\mathrm{CSIDH}}$	$\delta=1$	5	10	100
2HBC	1.0	4.67	9.26	91.76
OMTUP ($m_s=0$)	5.0	23.4	46.3	458.8
OMTUP ($m_s=8$)	21.0	98.1	194	1927

$\alpha_{\mathcal{U},\mathrm{CSIDH}}^{\mathrm{benchmark}}$	$\delta=1$	5	10	100
2HBC	0.971	4.59	8.84	91.9
OMTUP ($m_s=0$)	4.83	20.5	42.1	395
OMTUP ($m_s=8$)	19.1	80.5	170	1376

Table 2. In the left table are theoretical estimates for the cost reduction function $\alpha_{\mathcal{U},\mathrm{CSIDH}}(\delta, 5)$ for different δ in the 2HBC and OMTUP assumptions. We choose $m_r = 3$ and compare the cases $m_s = 0$ ($\beta = 75\%$) and $m_s = 8$ ($\beta = 99\%$). The right table represents benchmarks using the *Velusqrt* software package from [2]. We define $\alpha_{\mathcal{U},CSIDH}^{\mathrm{benchmark}}$ as the ratio between the number of CPU cycles the servers has to perform during the delegated protocol execution and the number of CPU cycles for a local computation. The benchmarks were done in Magma v2.25-6 on an Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz with 128 GB memory.

Communication cost. The communication cost of the full protocol is given by four times the costs established in equation (9), since CIso is invoked twice with two servers each time. The total costs are summarized in Table 3. 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.

	Upload					Download
	$\mathrm{Cl}(\mathcal{O})$	$\delta = 1$	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

Table 3. 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 $m_s = 8$ ($\beta = 99\%$).

CSIDH-1792. As a comparison to CSIDH-512, we also consider the larger parameter set for CSIDH-1792 proposed and analyzed in [4], with $\log_2 p \approx 1792$, $n = 209$, $B = 10$. We take the value $\lambda = 104$ from [4, Table 8] and find $k = 24$. We summarize our results on the left table of Table 4.

SQALE'd CSIDH-4096. We look at the SQALE'd CSIDH-4096 proposal from [9]. CSIDH-4096 uses $n = 417$, $\log_2 p \approx 4096$ and $B = 1$, which yields $\#\mathbb{B}(N) \approx 2^{661} \ll \#\mathrm{Cl}(\mathcal{O})$. Using $\lambda = 124$ as an estimate (cf. [9, Table 3]) yields $k = 40$. Our results are summarized on the right of Table 4. It is interesting to note that the gains are similar to the CSIDH-1792 case. 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.

$\alpha^{1792}_{\mathrm{CSIDH}}$	$\delta = 1$	5	10	100	$\to \infty$
2HBC	0.545	0.186	0.124	0.064	0.057
OMTUP	1.033	0.315	0.191	0.071	0.057

$\alpha^{4096}_{\mathrm{CSIDH}}$	$\delta = 1$	5	10	100	$\to \infty$
2HBC	0.548	0.215	0.139	0.058	0.048
OMTUP	1.048	0.381	0.230	0.068	0.048

Table 4. Theoretical estimates for different δ for the cost reduction functions $\alpha^{1792}_{\mathrm{CSIDH}} := \alpha_{\mathrm{CSIDH}}(\delta, 10, 209, 24, b)$ and $\alpha^{4096}_{\mathrm{CSIDH}} := \alpha_{\mathrm{CSIDH}}(\delta, 1, 417, 40, b)$, representing CSIDH-1792 from [4] and SQALE'd CSIDH-4096 from [9], respectively, in the 2HBC and OMTUP assumptions.

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 $\mathfrak{b}_1, \ldots, \mathfrak{b}_t$ 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 CIso (possibly in parallel). The delegator is left with computing the $\mathfrak{r}^*$-part of each of these delegations, we therefore find

$$\alpha_{\text{SeaSign}}(\delta, B, n, t, b) = (1+b)\frac{\mathsf{Ex}_{R^*}(\delta, (nt+1)B)}{\mathsf{Ex}_S((nt+1)B)},$$

choosing the same δ for each step. The instantiation in [13] again uses the parameter set from CSIDH-512 [7]. We show the cost reduction for different values of δ in Table 5. We consider the case $b = 0$ only, as $b = 1$ is simply double the value.

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 Table 6.

α_{SeaSign}	$\delta = 1$	5	10	100	$\to \infty$
2HBC	0.487	0.124	0.064	0.007	0.000
OMTUP	0.975	0.248	0.129	0.013	0.000

Table 5. Theoretical estimates of the cost reduction function $\alpha_{\text{SeaSign}}(\delta, 5, 74, t, b)$ for different δ. The cost difference between different $t \in \{1, \ldots, 128\}$ is negligible, so that the results hold for any of these choices.

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 $\mathrm{Cl}(\mathcal{O})$, the authors construct a signature scheme in the random oracle model based on the original identification protocol from Rostovtsev and Stolbunov [25, 28]. 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 $\mathfrak{b}_1, \ldots, \mathfrak{b}_t$, depending on the public key size 2^s. In

	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

Table 6. Communication cost of SeaSign in the CSIDH-512 parameter instantiation (assuming unknown $\mathrm{Cl}(\mathcal{O})$) in the 2HBC and OMTUP assumptions. We compare the cases $t = 32$ and $t = 128$, which depend on the public key size and the targeted security level.

contrast to SeaSign however, these elements can simply be sampled from $\mathbb{Z}_{\#\mathrm{Cl}(\mathcal{O})}$ 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 CIso, but knowing $\mathrm{Cl}(\mathcal{O})$ has now the advantage of not having to resort to ShrVec, and therefore not needing to compute the $\mathfrak{r}^*$ 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 $\mathbb{Z}_{\#\mathrm{Cl}(\mathcal{O})}$ 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.

7 Conclusion

This work presents a first approach of securely and verifiably delegating isogeny computations to potentially untrusted servers in the CSIDH setting. Delegation reduces the cost of intensive isogeny-based cryptographic protocols for computationally limited devices and thus presents a practical approach to large-scale deployment of post-quantum cryptographic schemes on mobile devices. We presented two algorithms and showed their application to different instances of CSIDH [7, 4, 9] as well as 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 11.7% and 4.7% of the cost of the local computation for CSIDH-512 and SQALE'd CSIDH-4096, respectively, while the cost of SeaSign quickly reduces to a few percent and asymptotically vanishes. Using the known class group of CSI-FiSh, its cost reduces to element generation in $\mathbb{Z}_{\#\mathrm{Cl}(\mathcal{O})}$.

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 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. Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, David Jao, Brian Koziel, Brian LaMacchia, Patrick Longa, et al. Supersingular isogeny key encapsulation. *Submission to the NIST Post-Quantum Standardization project*, 2017.
2. Daniel Bernstein, Luca De Feo, Antonin Leroux, and Benjamin Smith. Faster computation of isogenies of large prime degree. *arXiv preprint arXiv:2003.10118*, 2020.
3. Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. CSI-FiSh: efficient isogeny based signatures through class group computations. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 227–247. Springer, 2019.
4. Xavier Bonnetain and André Schrottenloher. Quantum security analysis of CSIDH. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 493–522. Springer, 2020.
5. Wouter Castryck and Thomas Decru. Csidh on the surface. In *International Conference on Post-Quantum Cryptography*, pages 111–129. Springer, 2020.
6. Wouter Castryck, Steven D Galbraith, and Reza Rezaeian Farashahi. Efficient arithmetic on elliptic curves using a mixed Edwards-Montgomery representation. *IACR Cryptol. ePrint Arch.*, 2008:218, 2008.
7. Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: an efficient post-quantum commutative group action. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 395–427. Springer, 2018.
8. Denis X Charles, Kristin E Lauter, and Eyal Z Goren. Cryptographic hash functions from expander graphs. *Journal of Cryptology*, 22(1):93–113, 2009.
9. Jorge Chávez-Saab, Jesús-Javier Chi-Domınguez, Samuel Jaques, and Francisco Rodrıguez-Henrıquez. The sqale of csidh: Square-root vélu quantum-resistant isogeny action with low exponents. Technical report, Cryptology ePrint Archive, Report 2020/1520, 2020. https://eprint. iacr. org . . . , 2020.
10. Céline Chevalier, Fabien Laguillaumie, and Damien Vergnaud. Privately outsourcing exponentiation to a single server: cryptanalysis and optimal constructions. In *European Symposium on Research in Computer Security*, pages 261–278. Springer, 2016.
11. Andrew Childs, David Jao, and Vladimir Soukharev. Constructing elliptic curve isogenies in quantum subexponential time. *Journal of Mathematical Cryptology*, 8(1):1–29, 2014.
12. Jean Marc Couveignes. Hard homogeneous spaces. *IACR Cryptol. ePrint Arch.*, 2006:291, 2006.
13. Luca De Feo and Steven D Galbraith. SeaSign: Compact isogeny signatures from class group actions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 759–789. Springer, 2019.
14. Lov K Grover. A fast quantum mechanical algorithm for database search. *arXiv preprint quant-ph/9605043*, 1996.

15. Susan Hohenberger and Anna Lysyanskaya. How to securely outsource cryptographic computations. In *Theory of Cryptography Conference*, pages 264–282. Springer, 2005.
16. David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *International Workshop on Post-Quantum Cryptography*, pages 19–34. Springer, 2011.
17. Thorsten Kleinjung. Quadratic sieving. *Mathematics of Computation*, 85(300):1861–1873, 2016.
18. Greg Kuperberg. A subexponential-time quantum algorithm for the dihedral hidden subgroup problem. *SIAM Journal on Computing*, 35(1):170–188, 2005.
19. Kohei Nakagawa, Hiroshi Onuki, Atsushi Takayasu, and Tsuyoshi Takagi. L1-norm ball for csidh: Optimal strategy for choosing the secret key space. *IACR Cryptol. ePrint Arch.*, 2020:181, 2020.
20. NIST. NIST post-quantum cryptography PQC, 2020. https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.
21. Robi Pedersen and Osmanbey Uzunkol. Secure delegation of isogeny computations and cryptographic applications. In *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop*, pages 29–42, 2019.
22. Robi Pedersen and Osmanbey Uzunkol. Delegating supersingular isogenies over $\mathbb{F}_{p^2}$ with cryptographic applications. volume 2021, page 506, 2021.
23. Chris Peikert. He gives C-sieves on the CSIDH. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 463–492. Springer, 2020.
24. Oded Regev. A subexponential time algorithm for the dihedral hidden subgroup problem with polynomial space. *arXiv preprint quant-ph/0406151*, 2004.
25. Alexander Rostovtsev and Anton Stolbunov. Public-key cryptosystem based on isogenies. *IACR Cryptol. ePrint Arch.*, 2006:145, 2006.
26. Carl Siegel. Über die classenzahl quadratischer zahlkörper. *Acta Arithmetica*, 1(1):83–86, 1935.
27. SIKE. Supersingular Isogeny Key Encapsulation, 2018. https://sike.org.
28. Anton Stolbunov. 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.
29. Anton Stolbunov. Cryptographic schemes based on isogenies. 2012.
30. Osmanbey Uzunkol, Jothi Rangasamy, and Lakshmi Kuppusamy. Hide the modulus: a secure non-interactive fully verifiable delegation scheme for modular exponentiations via CRT. In *International Conference on Information Security*, pages 250–267. Springer, 2018.
31. Jacques Vélu. Isogénies entre courbes elliptiques. *CR Acad. Sci. Paris, Séries A*, 273:305–347, 1971.