Secure Delegation of Isogeny Computations and Cryptographic Applications[⋆]

Robi Pedersen[1] and Osmanbey Uzunkol[1,2]

[1] Fernuniversität in Hagen, Hagen, Germany
[2] Flensburg University of Applied Sciences, Flensburg, Germany

robi.pedersen@protonmail.com, osmanbey.uzunkol@gmail.com

Abstract. We address the problem of speeding up isogeny computation for supersingular elliptic curves over finite fields using untrusted computational resources like third party servers or cloud service providers (CSPs). We first propose new, efficient and secure delegation schemes. This especially enables resource-constrained devices (e.g. smart cards, RFID tags, tiny sensor nodes) to effectively deploy post-quantum isogeny-based cryptographic protocols. To the best of our knowledge, these new schemes are the first attempt to generalize the classical secure delegation schemes for group exponentiations and pairing computation to an isogeny-based post-quantum setting. Then, we apply these secure delegation subroutines to improve the performance of supersingular isogeny-based zero-knowledge proofs of identity. Our experimental results show that, at the $128-$bit quantum-security level, the proving party only needs about 3% of the original protocol cost, while the verifying party's effort is fully reduced to comparison operations. Lastly, we also apply our delegation schemes to decrease the computational cost of the decryption step for the NIST postquantum standardization candidate SIKE.

1 Introduction

It is a well-known fact that currently deployed public-key cryptographic primitives will be totally insecure against the efficient implementation of Shor's algorithm on large scale quantum computers [32]. This has resulted in the ever increasing requirement for new candidates for quantum-resistant cryptographic primitives, i.e. digital signatures, public-key encryption and key-establishment algorithms and more advanced practical cryptographic protocols like zero-knowledge proofs and oblivious transfers. As a consequence, at the end of 2017, the United States' National Institute of Standards and Technology (NIST) initiated a standardization project with an initial call for proposals for post-quantum public-key cryptography. 17 out of 59 public-key encryption and key-establishment algorithms have achieved to be selected for the second round in January 2019 [27]. The intractability of chosen public-key encryption and key-establishment algorithms is based on the hardness of either lattice-based, code-based, or supersingular elliptic curve isogeny assumptions.

Elliptic curve isogeny assumptions were first used by Stolbunov [36] to propose quantum-resistant isogeny-based Diffie-Hellman key agreement protocols by rediscovering the original work of Couveignes related to the hard homogeneous spaces [11]. However, Childs et al. proposed a quantum algorithm that can extract private keys in subexponential time [7], if the underlying elliptic curve is ordinary, i.e. if the endomorphism ring of the curve is commutative. Jao and De Feo [21] then proposed a quantum-resistant isogeny-based Diffie-Hellman key agreement protocol (SIDH) using supersingular elliptic curves over finite fields. In the same paper, Jao and De Feo propose a supersingular isogeny-based public key encryption scheme as well as a zero-knowledge proof identification scheme. Based on SIDH, a key encapsulation scheme SIKE [33] is submitted as a candidate to the

⋆ This is a preprint of an article, which will be published in the proceedings of the 2019 ACM Cloud Computing Security Workshop (CCSW'19). The final authenticated version will be available online at https://doi.org/10.1145/3338466.3358913.

NIST standardization project. SIKE is the single candidate among the above 17 algorithms whose security assumption is based on a supersingular elliptic curve isogeny assumption, i.e. on the difficulty of computing isogenies between elliptic curves over finite fields. In particular, the security assumption of the schemes in [21] rely on the pseudo-random walks on isogeny graphs of suitable elliptic curves over finite fields, and the property that (unlike ordinary elliptic curves) a supersingular elliptic curve over a finite field $\mathbb{F}_q$ has a non-abelian endomorphism ring. The latter ensures in particular that the above-mentioned quantum attacks, which compute the isogenies of elliptic curves in subexponential time, are not applicable to supersingular elliptic curves. Another interesting application of isogeny-based cryptography is the post-quantum resistant zero-knowledge proof of identity scheme [21].

The supersingular elliptic curve isogeny problem has also recently been used in several new cryptographic applications including provably secure hash functions [4] and digital signatures [40]. Very recently, with its immediate applications in designing random beacons and consensus from proof-of-resources (aiming to reduce the energy consumption of blockchains based on proof-of-work), it is also used to construct verifiable delay functions, which take a prescribed wall-clock time to evaluate a function f in not less than T sequential steps to produce an output $y = f(x)$ for a given input x [16].

Motivation. Although cryptographic protocols using supersingular isogeny-based security assumptions require a very small key length when compared to other post-quantum protocols, these have the main disadvantage of being relatively slow, even after a substantial amount of recent research results addressing to improve their performance [8, 20, 28]. For instance, zero-knowledge proofs require repeated protocol executions and have hence been described as *a purely theoretical and pedagogical* application of isogeny-based cryptography in [21]. Moreover, as also highlighted by NIST [27], post-quantum algorithms and protocols are highly required to efficiently work on resource-constrained devices with limited processing power, storage and battery life, such as smart cards, RFID cards and tiny sensors in order to realize quantum-resistant lightweight cryptographic primitives, e.g. in the context of the Internet of Things (IoT).

A possible practical approach to reduce the cost of expensive computations of cryptographic primitives is to delegate their computation to more powerful (but potentially malicious) third party servers including cloud service providers (CSPs) in a secure and verifiable manner. Starting with the first proposal of Hohenberger and Lysyanskaya [19], several schemes for secure and verifiable delegation of both group exponentiations and pairing computations have been recently proposed. These schemes help to design practically deployable, lightweight privacy- and anonymity-oriented cryptographic protocols by reducing the resource-consuming expensive arithmetical operations for cryptographic algorithms like group exponentiations and pairing computation, see for instance [19, 22, 37] and their references.

Our Contribution. The first contribution of this paper is to extend the classical result for secure delegation of group exponentiation and pairing computation to the secure delegation of isogeny computation. To the best of our knowledge, it is the first attempt which extends the classical delegation schemes of cryptographic computation to the delegation of quantum-resistant cryptographic primitives. Specifically, we propose two delegation algorithms for isogeny computation which are efficient with very high verifiability probabilities using the security model of Hohenberger and Lysyanskaya [19]. In particular,

1. We adapt the existing precomputation techniques [3, 26, 38] to the generation of elliptic curve torsion-points for use in the main steps of our delegation algorithms.
2. We propose the *public isogeny and scalar multiplication algorithm* ScIso, which takes as input a *public* supersingular elliptic curve E, a *public* kernel generator A, a *hidden* set of scalar factors and three sets of elliptic curve points. The algorithm then outputs the map of these points via

the kernel generated by A, while the delegator can optionally hide points or multiply them with the scalar factors.

3. We propose the *hidden isogeny algorithm* HIso, which takes as input a *public* supersingular elliptic curve E, a *hidden* scalar a and two *public* basis points P_A, Q_A, then produces the *hidden* output $E/\langle A\rangle$, where $A = P_A + [a]Q_A$.

Our second contribution is to use these delegation algorithms in order to improve the efficiency of isogeny-based cryptographic applications. In particular, we improve the efficiency of the zero-knowledge proof of identity protocol (ZKPI) presented in [21] and show, how our algorithms can be used to delegate the shared key establishment step in the supersingular isogeny Diffie-Hellman key exchange protocol (SIDH) as well as the decryption step within the public-key encryption scheme from [21] and within the supersingular isogeny key encapsulation algorithm (SIKE) [1, 33]. We effectively reduce the computational cost of a round in the ZKPI to about 3% of the original cost over the finite field $\mathbb{F}_{p^2}$ of characteristic $p_{751} = 2^{372}3^{239} - 1$, while our contribution to the other algorithms yields a gain factor of almost 2. We further show that the gain increases with larger security p.

Our last contribution is the implementation of the new delegation schemes using the PQCrypto-SIDH v3.0 library [23] for elliptic curves over the finite fields $\mathbb{F}_{p^2}$ with characteristics $p_{503} = 2^{250} \cdot 3^{159} - 1$ and $p_{751} = 2^{372} \cdot 3^{239} - 1$, corresponding to at least 80- and 128-bit quantum security levels, respectively. We compare the local computation of the protocols with the delegated computation using our algorithms. In particular, these experiments show that the higher the security level will be, the better and more visible the advantages of using delegated isogeny computations become for resource-constrained devices when compared with their local computation.

2 Preliminaries and security model

In this section, we introduce our notation and some basic properties of isogeny-based cryptography and revisit the supersingular zero-knowledge proof of identity protocol introduced in [21]. Afterwards, we give an overview of computational costs of common operations required in isogeny-based cryptographic protocols. Finally, we will revisit the secure delegation model proposed in [19] and end this section by outlining the required steps to securely delegate isogeny computations.

2.1 Basic notation and terminology

We denote by $\mathbb{Z}_k^*$ the multiplicative group of integers modulo k, where $\mathbb{Z}_k := \mathbb{Z}/k\mathbb{Z},\ k \in \mathbb{N}^{>1}$. Further, let $\mathbb{F}_q$ be a finite field of order q with characteristic $p = \mathrm{char}(\mathbb{F}_q)$. Elliptic curves defined over $\mathbb{F}_q$ will be written as $E/\mathbb{F}_q$, or simply as E. Points on E are indicated as capital Latin letters, while $\mathcal{O}$ denotes the distinct point at infinity. The m-fold successive addition of a point P with itself is called the *multiplication-by-m* map $[m] : P \mapsto [m]P$, where $m \in \mathbb{Z}$, and is also referred to as the scalar multiplication of P with m. The kernel of $[m]$ is the set of all points of order dividing m, i.e. for which it holds that $[m]P = \mathcal{O}$. These kernels define subgroups, referred to as *torsion groups* of E. The m-torsion group is written as $E[m]$. Isogenies define epimorphisms between elliptic curves and are denoted by lowercase Greek letters, such as ϕ, α or β. Further, we will denote by $\langle A\rangle$, the set generated by a point $A \in E$ and by $\langle P, Q\rangle$ the set generated by all possible linear combinations $[m]P + [n]Q$ of P and Q, where $m, n \in \mathbb{Z}$. If P and Q are clear from the context, we will also use the shorthand notation $A = [a_1]P + [a_2]Q =: (a_1, a_2)$. Finally, we denote the codomain of an isogeny ϕ with kernel $\ker\phi = G$, where G is a group, by E/G. If $G = \langle A\rangle$, we also write $E_A := E/\langle A\rangle$.

2.2 Supersingular Isogeny Cryptography

The cryptographic systems from [21] are all based on supersingular elliptic curves defined over finite fields $\mathbb{F}_{p^2}$, where $p = l_A^{e_A} l_B^{e_B} f \mp 1$. l_A and l_B are small, distinct primes and are usually chosen to be

2 and 3, while f is a small co-factor that ensures that p is prime. Choosing p in such a way ensures that $E/\mathbb{F}_{p^2}$ has two large torsion subgroups $E[l_A^{e_A}]$ and $E[l_B^{e_B}]$. These torsion groups can each be generated by two linear independent basis points $P_i, Q_i \in E[l_i^{e_i}]$ and consist of $l_i^{e_i-1}(l_i+1)$ distinct cyclic subgroups of order l_i for $i \in \{A, B\}$. Each of these cyclic subgroups can be generated by a linear combination of the basis points, e.g. $\langle [m]P_i + [n]Q_i \rangle$ where at least one of n, m is necessarily coprime to l_i. Every cyclic subgroup also defines the unique kernel of a separable isogeny of smooth degree $l_i^{e_i}$ [34], which in turn defines a path through the isogeny graph to another supersingular elliptic curve.

The advantage of having isogenies of smooth degree is that they can be decomposed into their prime factors, e.g. an isogeny ϕ of degree l^e, or an l^e-isogeny for short, can be decomposed into a chain of e consecutive l-isogenies $\phi_{e-1} \circ \cdots \circ \phi_0$, enabling an efficient computation of isogenies. At every step, we have to map the initial kernel generator along, as it will be required to generate the next one. When computing chains of this type, there is a trade-off between scalar multiplications and isogeny computations. The authors in [21] argue, that an *optimal* trade-off strategy has about $\frac{e}{2}\log_2 e$ of each with a slight bias towards the cheaper of the two. The case where both occur equally is called the *balanced* scenario, and only has a performance loss of a few percent to the optimal case.

The computation of isogenies, and generally the arithmetic on elliptic curves, at least for our purposes, can be greatly improved in terms of efficiency by working with Montgomery curves [10, 24]. The general approach is to work in projective coordinates, in which Montgomery curves are defined by the equation

$$E : bY^2Z = X^3 + aX^2Z + XZ^2, \quad \text{with} \quad j(E) = \frac{256(a^2-3)^3}{a^2-4} \tag{1}$$

their j-invariant, and points can be expressed on the Kummer line $\mathbb{P}^1$ using only two coordinates, e.g. $P = (X : Z)$. Interestingly, like the j-invariant, neither isogeny computations, nor scalar multiplications or additions need the curve parameter b, so we do not need to compute it along the chain of isogenies. Costs of operations on Montgomery curves can be found in [2, 9, 10, 21, 24, 29] and are revisited in Section 2.4.

Supersingular isogeny zero-knowledge proof of identity. The protocol in [21] revolves around the following commutative diagram:

$$\begin{tikzcd} E \arrow[r, "\alpha"] \arrow[d, "\beta"'] & E_A \arrow[d, "\beta'"] \\ E_B \arrow[r, "\alpha'"] & E_{AB} \end{tikzcd}$$

Let $E/\mathbb{F}_{p^2}$ be a supersingular elliptic curve with $p = l_A^{e_A} l_B^{e_B} \mp 1$ and let $\langle P_A, Q_A \rangle = E[l_A^{e_A}]$ and $\langle P_B, Q_B \rangle = E[l_B^{e_B}]$ be torsion group generators. A proving party, usually called Peggy, wants to convince a verifying party, Victor, that she knows a generator of the isogeny $\alpha : E \to E_A := E/\langle A \rangle$ without revealing it. The pair (E, E_A) represents Peggy's public key and at each round, she publishes new, random $E_B := E/\langle B \rangle$ and $E_{AB} := E/\langle A, B \rangle$. Depending on a challenge-bit b by Victor, she then either reveals the kernel generators $(B, \alpha(B))$ of β and β' or the kernel generator $(\beta(A))$ of α', for $b = 0$ or $b = 1$, respectively. Victor verifies, if these generators actually generate the correct kernel isogenies.

2.3 Equivalence relation

Let $\langle P, Q\rangle = E[l^e]$ and let $R = (r_1, r_2)$ and $S = (s_1, s_2)$ be points in $E[l^e]$. We define the equivalence relation [17]

$$R \sim S \Leftrightarrow \exists \lambda \in \mathbb{Z}_{l^e}^* : R = [\lambda]S, \text{or}$$
$$(r_1, r_2) \sim (s_1, s_2) \Leftrightarrow \exists \lambda \in \mathbb{Z}_{l^e}^* : (r_1, r_2) = [\lambda](s_1, s_2) = (\lambda s_1, \lambda s_2).$$

which implies that $\langle R\rangle = \langle S\rangle$ and therefore defines the following equivalence class

$$[R] = \{[\lambda]R \mid \lambda \in \mathbb{Z}_{l^e}^*\}$$

As a consequence, we introduce the following normal forms[3]

- If $r_1 \in \mathbb{Z}_{l^e}^*$, choose $\lambda = r_1^{-1}$, so that $R \sim (1, r_1^{-1}r_2) =: (1, r)$.
- Otherwise, if $r_2 \in \mathbb{Z}_{l^e}^*$, choose $\lambda = r_2^{-1}$, so that $R \sim (r_1 r_2^{-1}, 1)$.

Note that for any given basis P, Q and point $R = (r_1, r_2) \in E[l^e]$ it is possible to extract the factors r_1 and r_2 using pairings, i.e. let $e : E \times E \to G$ be a pairing and G a group, then we can compute

$$e(R, Q) = e([r_1]P + [r_2]Q, Q) = e([r_1]P, Q),$$
$$e(P, R) = e(P, [r_1]P + [r_2]Q) = e(P, [r_2]Q),$$

and extract the factors r_1 and r_2 by computing the discrete logarithm, which is easy on supersingular elliptic curves using the Pohlig-Hellman algorithm [39, 35]. In particular, we can always extract the normal form by computing the factor $r = r_1^{-1}r_2$, if it exists.

2.4 Computational costs

We define by m, s and i the computational cost of multiplication, squaring and inversion over $\mathbb{F}_{p^2}$, respectively. As in [2, 21], we neglect the cost of comparisons, as well as additions and subtractions on $\mathbb{F}_{p^2}$, and assume that the operations on $\mathbb{F}_{p^2}$ satisfy $s = 0.8m$ and $i = 100m$. On Montgomery curves in projective coordinates, we find $A = 3m + 2s = 4.6m$ and $D = 2m + 2s = 3.6m$, for the cost of point (pseudo-)addition A and (pseudo-)doubling D [2, 9, 10, 21, 24, 29].

Let $S(k)$ define the cost of a scalar multiplication by a factor k on E. We have $S(2) = D = 3.6m$ and $S(3) = D+A = 8.2m$. For larger scalar multiplications, we use the Montgomery ladder algorithm [24], which costs $M = 8.2m$ at each step in the *for*-loop, yielding the total [2, 10, 15]

$$S(k) = M\lceil \log_2 k\rceil - A. \tag{2}$$

We further define by $I(k)$ the computational cost of a k-isogeny. Isogeny computations between Montgomery curves are usually composed of two parts, i.e. finding the curve parameter a of the codomain and mapping a point to the new curve. We define by $I_a(k)$ the former and by $I_P(k)$ the latter, while $I(k)$ designates the total cost. Using the results from [9, 29], we get

$$\begin{aligned} I_P(2) &= 4m, & I_P(3) &= 4m + 2s = 5.6m, \\ I_a(2) &= 2s = 1.6m, & I_a(3) &= 4m + 2s = 5.6m, \\ I(2) &= 4m + 2s = 5.6m, & I(3) &= 8m + 4s = 11.2m \end{aligned} \tag{3}$$

[3] Working with isogenies, we can always assume that either $r_1 \in \mathbb{Z}_{l^e}^*$ or $r_2 \in \mathbb{Z}_{l^e}^*$, so that R is of full order. Otherwise the isogeny generated by R would not be of maximal degree, reducing the overall security.

For larger isogenies of smooth degree l^e, we can use the strategy described in [21] and decompose the isogeny into a chain of l-isogenies, the chain being of length e. At each step of the chain, the kernel generator has to be mapped along to be able to compute the next step. The cost of the balanced scenario, as explained in Section 2.2 can be written as

$$I(l^e) = (I(l) + S(l))\frac{e}{2}\log_2 e, \tag{4}$$

while the optimal cost deviates by a small percentage from this result [21].

2.5 Communication costs

In order to be able to better estimate the communication cost during delegation, we denote by $c(p)$ the amount of information required for an element in $\mathbb{Z}_p$, where it holds that

$$c(p) \leq 1 + \lfloor \log_2 p \rfloor.$$

In the table below, we indicate the information of different elements needed throughout this work. Following the discussion in Section 2.2, elliptic curves can be described by their curve parameter a only, while points on elliptic curves can be described by two coordinates $(X : Z)$, or even one, when Z is scaled to 1.

element in $\mathbb{Z}_p$ or $\mathbb{F}_p$:	$c(p)$,
element in $\mathbb{F}_{p^2}$:	$2c(p)$,
point on $E/\mathbb{F}_{p^2}$:	$4c(p)$; $2c(p)$ if $Z = 1$,
elliptic curve $E/\mathbb{F}_{p^2}$:	$2c(p)$.

2.6 Security assumptions and model

In this section, we revisit the security model for delegated cryptographic computations proposed in [19]. Delegation algorithms are split into two parties, a trusted component $\mathcal{T}$ and an untrusted server $\mathcal{U}$ (or multiple servers $\mathcal{U}_1, \ldots, \mathcal{U}_n$), to which $\mathcal{T}$ can make oracle queries. The idea is, that the joint effort of $\mathcal{T}$ and $\mathcal{U}$, denoted by $\mathcal{T}^{\mathcal{U}}$, implements an algorithm Alg, for which $\mathcal{T}$ has much less computational cost than executing Alg alone. The downside is, that a potentially malicious server $\mathcal{U}'$ may try to either (1) extract confidential information from its interaction with $\mathcal{T}$, or (2) return wrong results to $\mathcal{T}$'s queries. Thus $\mathcal{T}$ needs (1) to make sure, that $\mathcal{U}'$ cannot obtain any useful information, and (2) to verify if the outputs given by $\mathcal{U}'$ are actually correct. Another adversary considered in [19] is the environment $\mathcal{E}$, representing any possible third party, including the provider/manufacturer of $\mathcal{U}$ or $\mathcal{U}'$. The full adversary can thus be modeled as the pair $\mathcal{A} = (\mathcal{E}, \mathcal{U}')$. A key assumption of this model is that before $\mathcal{T}$ starts to use $\mathcal{U}'$, the adversaries can devise a joint strategy, but when $\mathcal{T}$ starts using $\mathcal{U}'$, there will not be a direct communication channel between either $\mathcal{E}$ and $\mathcal{U}'$, nor between the different $\mathcal{U}'_1, \ldots, \mathcal{U}'_n$. These considerations imply that the three parties, $\mathcal{T}$, $\mathcal{E}$ and $\mathcal{U}$, have distinct views and/or access to the inputs and outputs of Alg. In that sense, we distinguish *secret* (hidden from $\mathcal{U}$ and $\mathcal{E}$), *protected* (hidden from $\mathcal{U}$) and *unprotected* (each entity has access) in- and outputs. The protected and unprotected inputs can further be subdivided into *honest* and *adversarial*, depending on whether the inputs originate from a trusted source or not. An algorithm Alg with exactly these five inputs and three outputs is said to obey the *outsource input/output specification* (or *outsource-IO*) [19].

$\mathcal{E}$ and $\mathcal{U}'$ could try to establish an indirect communication channel between them via the unprotected inputs and outputs of Alg, e.g. $\mathcal{E}$ could send instructions to $\mathcal{U}'$ via the adversarial, unprotected input, and $\mathcal{U}'$ could send gained information to $\mathcal{E}$ via the unprotected output. The first of these attempts poses an enormous security threat to the integrity of $\mathcal{U}'$, and $\mathcal{T}$ should therefore ensure that the adversarial, unprotected input stays empty. To circumvent the second problem, $\mathcal{T}$ has to make sure that both $\mathcal{U}'$ and $\mathcal{E}$ do not gain any useful information from $\mathcal{T}$'s interaction with $\mathcal{U}$. This is formalized in the following definition:

Definition 1 (Outsource-security). *[19] Let $Alg(\cdot,\cdot,\cdot,\cdot,\cdot)$ be an algorithm with outsource-IO. A pair of algorithms $(T,\mathcal{U})$ is said to be an* outsource-secure *implementation of Alg if:*

- ***Correctness**: $\mathcal{T}^{\mathcal{U}}$ is a correct implementation of Alg.*
- ***Security**: For all probabilistic polynomial-time adversaries $\mathcal{A} = (\mathcal{E},\mathcal{U}')$, there exist probabilistic expected polynomial-time simulators $(\mathcal{S}_1, \mathcal{S}_2)$, such that the following pairs of random variables are computationally indistinguishable. We assume that the honestly-generated inputs are chosen by a process I.*
  - ***Pair One**: $\mathcal{E}VIEW_{real} \sim \mathcal{E}VIEW_{ideal}$ (The external adversary $\mathcal{E}$ learns nothing.)*
  - ***Pair Two**: $\mathcal{U}VIEW_{real} \sim \mathcal{U}VIEW_{ideal}$ (The untrusted software $\mathcal{U}'$ learns nothing.)*

  *The details of the experiments **Pair One** and **Pair Two** can be found in Definition 2.2 of [19]. Note that if $\mathcal{U}' = (\mathcal{U}'_1, \ldots, \mathcal{U}'_n)$ consists of multiple servers, then there is a PPT-simulator $\mathcal{S}_{2,i}$ for each of their views.*

We further formalize $\mathcal{T}$'s efficiency gain and ability to verify the returned outputs during its interaction with $\mathcal{U}'$ in the following definitions:

Definition 2 (α-efficiency). *[19] A pair of algorithms $(T,\mathcal{U})$ are an α-efficient implementation of an algorithm Alg if*

- *$(T,\mathcal{U})$ are an outsource-secure implementation of Alg, and*
- *for all inputs x, the running time of $\mathcal{T}$ is smaller than the running time of $Alg(x)$ by at least a factor α.*

Definition 3 (β-checkability). *[19] A pair of algorithms $(T,\mathcal{U})$ are a β-checkable implementation of an algorithm Alg if*

- *$(T,\mathcal{U})$ are an outsource-secure implementation of Alg, and*
- *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 $\geq \beta$.*

Definition 4 ((α, β)-outsource-security). *[19] A pair of algorithms $(T,\mathcal{U})$ are an (α, β)-outsource secure implementation of an algorithm Alg, if they are both α-efficient and β-checkable.*

There are different paradigms of delegation models, which differ along the quantity of servers and their intentions, ranging from behaving honest, honest-but-curious to malicious.

The models we will analyze in this work are the following:

Definition 5 (Honest-but-curious). *[6] The* one honest-but-curious program model *defines the adversary as $\mathcal{A} = (\mathcal{E},\mathcal{U}')$ and assumes, that $\mathcal{U}'$ returns correct results, but may try to extract useful data.*

Definition 6 (OMTUP). *[5, 19, 37] In the* one-malicious version of a two untrusted program model, *the adversary is given by $\mathcal{A} = (\mathcal{E}, (\mathcal{U}'_1,\mathcal{U}'_2))$. It is assumed 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.*

2.7 Steps of the delegation of isogeny computations

Securely delegating isogeny computations first needs a (1) *precomputation step*, which we will call GenP, that allows $\mathcal{T}$ to generate (pseudo-)random pairs

$$(\kappa_1, \kappa_2, K) \in \mathbb{Z}_{l^e} \times \mathbb{Z}_{l^e} \times E[l^e],$$

where $K = [\kappa_1]P+[\kappa_2]Q$ and $\langle P, Q\rangle = E[l^e]$. One possibility is to assume that $\mathcal{T}$ either is intrinsically able to generate those pairs or that it has a trusted server that generates them in its place. Otherwise $\mathcal{T}$ can use adapted versions of preprocessing techniques similar to those in [3, 13, 26, 30, 31, 38] for the generation of elliptic curve torsion points. We will present these techniques in Section 3.

Using the tuples (κ_1, κ_2, K) from GenP, $\mathcal{T}$ can generate new, ephemeral keys, which it can use within the cryptographic protocols, but also new, random points. $\mathcal{T}$ can use these random points in order to (2) *shroud* some of the secret or protected inputs, before making (3) *queries* to the servers. For our intents and purposes, the servers $\mathcal{U}_i$ have the advertised functionality of taking an elliptic curve E and a kernel generator $A \in E$, as well as a set of points $\{P_1, \ldots, P_r\} \subset E$ and a set of pairs $\{(a_1, Q_1), \ldots, (a_s, Q_s)\} \subseteq \mathbb{Z} \times E$ as inputs, then computing the isogeny $\alpha : E \to E_A := E/\langle A\rangle$ and returning the codomain curve E_A and the maps $\{\alpha(P_1), \ldots, \alpha(P_r), [a_1]\alpha(Q_1), \ldots, [a_r]\alpha(Q_r)\}$ of the input points. We write:

$$\begin{aligned}&\mathcal{U}_i(E, A;\ \{P_1, \ldots, P_{n_P}\}, \{(a_1, Q_1), \ldots, (a_s, Q_s)\})\\ &\quad \leftarrow (E_A, \{\alpha(P_1), \ldots, \alpha(P_r)\}, \{[a_1]\alpha(Q_1), \ldots, [a_s]\alpha(Q_s)\}).\end{aligned}$$

If the elliptic curves and the kernel generator are clear from the context, we omit them from the notation. After the delegation, $\mathcal{T}$ has to (4) *verify*, if the outputs from the servers are correct, by checking if they fulfill a specific verification condition. To that end, $\mathcal{T}$ de-randomizes the outputs using information that only itself knows. If the verification fails, $\mathcal{T}$ rejects the outputs, otherwise, $\mathcal{T}$ can (5) *recover* the result from these outputs and its own, undisclosed information. Note that the verification step is not necessary under the honest-but-curious assumption from Definition 5.

3 Precomputation

In this section, we will present how the precomputation techniques from [3, 13, 26, 30, 31, 38] are applicable to the generation of elliptic curve torsion-points. Because of the structure of the underlying elliptic curves, 2^{e_2}- and 3^{e_3}-torsion points are required to be generated at the precomputation step. Since each of these torsion groups is generated by two basis points, it suggests itself to create our lookup-tables as follows.

Preprocessing step. Let $\langle P, Q\rangle \in E[l^e]$. Compute $P_i = [l^i]P$ and $Q_i = [l^i]Q$ for each $i \in \{0, \ldots, e-1\}$, and store (i, P_i) and (i, Q_i) in a table.

Because l^e-torsion points are stored in base l, the generation of 2^{e_2}-torsion points is slightly different from the generation of 3^{e_3}-torsion points, the former being generated similar to the BPV-generator in [3], while the generation of the latter is loosely based on the EBPV-generator in [26]. In analogy to these sources, we introduce a security parameter h defining the subset size, from which $\mathcal{T}$ generates the new point.

Invocation step for 2^{e_2}-torsion points. Generate two random subsets $S, S' \subseteq \{0, \ldots, e_2 - 1\}$ with $|S| = |S'| = h$. Let $\kappa = (\kappa_{e_2-1} \ldots \kappa_0)_2$ and $\kappa' = (\kappa'_{e_2-1} \ldots \kappa'_0)_2$ be two scalars, where $\kappa_i^{(')} = 1$ if $i \in S^{(')}$ and $\kappa_i^{(')} = 0$ otherwise. Then compute

$$K = \sum_{i\in S} P_i + \sum_{i\in S'} Q_i$$

and return (κ, κ', K).

Invocation step for 3^{e_3}-torsion points. Generate two random subsets $S, S' \subseteq \{0, \ldots, e_3 - 1\}$ with $|S| = |S'| = h$. Then, generate two random subsets $S_2 \subseteq S$ and $S_2' \subseteq S'$ and define $S_1 := S\backslash S_2$ and $S_1' := S\backslash S_2'$. Let $\kappa^{(')} = (\kappa^{(')}_{e_3-1} \ldots \kappa^{(')}_0)_3$ where $\kappa_i^{(')} = 2$ if $i \in S_2^{(')}$, $\kappa_i^{(')} = 1$ if $i \in S_1^{(')}$, and

$\kappa_i^{(')} = 0$, otherwise. Then compute

$$K = \left(\sum_{i\in S_1} P_i + \sum_{i\in S_1'} Q_i\right) + [2]\left(\sum_{i\in S_2} P_i + \sum_{i\in S_2'} Q_i\right)$$

and return (κ, κ', K).

We denote the respective invocation step on an elliptic curve E as $\mathsf{GenP}_E \leftarrow K = (\kappa, \kappa')$. We also assume that $\mathcal{T}$ is able to choose κ and κ' explicitly in order to perform a *directed point generation*, which we will denote by $\mathsf{GenP}_E(\kappa, \kappa') \leftarrow K = (\kappa, \kappa')$. The subscript E can be dropped if the underlying elliptic curve is clear from context.

The choice of working in base 2 and 3 allows $\mathcal{T}$ to have control over the order of the generated point, e.g. $\mathcal{T}$ can generate an l^{e-k}-torsion element, for an integer $k < e$, by ensuring that the last k entries of κ and κ' in base l are all zero, $(\kappa_{k-1}^{(')} \dots \kappa_0^{(')})_l = (0 \dots 0)_l$, while for full order points, T has to ensure that $\kappa_0 \neq 0$ or $\kappa_0' \neq 0$.

Since our point generation schemes are based on the schemes presented in [3] and [26], we assume the security to be similar to the considerations presented in these sources. We can easily extend the invocation steps to return truly random points, similar to the SMBL scheme proposed in [38], by choosing $\kappa, \kappa' \in \mathbb{Z}_{l^e}$ as truly random points in base l and then continuing the corresponding invocation steps.

We turn our attention to the computational and spatial costs of the point generation schemes in this section. We denote by $G_l(h)$ the cost of invoking the upper schemes, for $l = 2$ or $l = 3$, respectively, h representing the choice of the security parameter.

We find

$$G_l(h) = (2h-1)A + (l-2)D. \tag{5}$$

If we want to generate points in the standard form, this reduces to

$$G_l(h) = hA + (l-2)D. \tag{6}$$

For the truly random scheme, we can use the same results and plug in $h = e_l$ for the worst case and $h = e_l/2$ for the average case complexity.[4]

The tables' storage cost amounts to $2e_l$ pairs per torsion group. If $\mathcal{T}$ needs both tables, then $2(e_2+e_3)$ pairs are required to be stored. We can easily reduce the storage by increasing the number of operations during the invocation steps, e.g. if we want to reduce the size of the table by a factor f, we observe that the time complexity increases to

$$G_l^{(f)}(h) = G_l^{(1)}(h) + (f-1)(D + (l-2)S(3)). \tag{7}$$

This time-space trade-off is depicted in Figure 1. We can see that even reducing the table size to a tenth of the full size only increases the computational cost by about 11%. This is further reduced for larger h. The parameter f can thus easily be chosen according to the available space of the underlying hardware.

[4] A subset size of $h = 64$ in [3] would correspond to $h = 32$ in our case, since the subset size equals $|S| + |S'| = 2h$. However, depending on the case, it suffices to generate points in their standard form, so that the subset size reduces to h, and that we would need to choose $h = 64$ to yield the same security. In both cases, we have approximately the same cost, so that we will restrict our analysis to the former case.

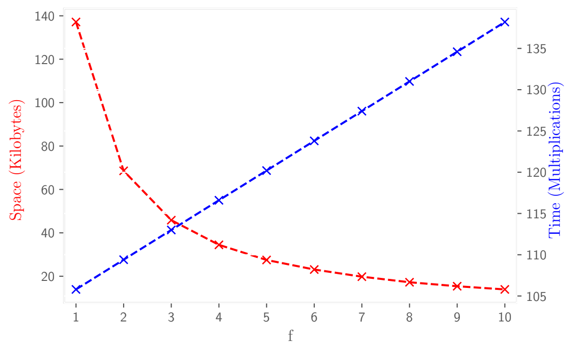


Fig. 1. Spatial cost (in red) of lookup-tables versus invocation cost (in blue) for different table sizes in terms of the table reduction factor f for $l = 2$ and $h = 32$.

4 SclIso: A delegation scheme for isogeny computations

In this section, we present a delegation algorithm, ScIso for isogeny computations which further includes scalar multiplication.

In contrast to delegation schemes related to the computation of modular exponentiation, the input and output domains of an isogeny computation are different. Our choice for the honest-but-curious and OMTUP assumptions simply comes from the need to verify if the target curve of an isogeny is correct. The problem however is that the delegator $\mathcal{T}$ would need to compute the isogeny itself, eliminating the need for delegation. Furthermore, all of the current modular exponentiation algorithms [5, 6, 19, 22, 25, 37, 38] use the fact, that the delegator $\mathcal{T}$ has already generated part of the solution itself, i.e. a factor that is never transmitted to $\mathcal{U}'$, but still used as part of the computation of the final result. In our case, not knowing the target curve beforehand, would only leave $\mathcal{T}$ the option to construct linear combinations from the results of the delegation procedure. However, if the malicious server(s) would consistently scale all results by a (previously determined) factor λ, coprime to the order of the points,[5] then there would be no way for $\mathcal{T}$ to detect that they are incorrect. We can avoid this problem by ensuring that the auxiliary server is honest, i.e. returns a correct curve, or in the case where we have two servers, that at least one consistently returns the correct curve, so that we can simply check, if both curves match. The honest-but-curious and OMTUP assumptions from Definitions 5 and 6 respectively exactly match those criteria.

Definition 7 (The ScIso-algorithm). *The* public isogeny and scalar multiplication algorithm ScIso *takes as inputs*

- *a supersingular elliptic curve* $E/\mathbb{F}_{p^2}$*, with* $p = l_A^{e_A} l_B^{e_B} f \mp 1$,
- *an isogeny kernel generator* $A \in E[l_A^{e_A}]$,
- *a set of points* $\{P_1, \ldots, P_r\} \subseteq E[l_A^{e_A}] \cup E[l_B^{e_B}]$*, where* $r \in \mathbb{N}$,
- *a set of points* $\{H_1, \ldots, H_u\} \subseteq E[l_A^{e_A}] \cup E[l_B^{e_B}]$*, where* $u \in \mathbb{N}$,
- *a set of pairs* $\{(a_1, Q_1), \ldots, (a_s, Q_s)\} \subseteq \mathbb{Z}_{l_A^{e_A}} \times (E[l_A^{e_A}] \cup E[l_B^{e_B}])$*, where* $s \in \mathbb{N}$,

ScIso *then produces the outputs*

- $E_A := E/\langle A \rangle$,
- $\Phi_P := \{\alpha(P_1), \ldots, \alpha(P_r)\}$,
- $\Phi_H := \{\alpha(H_1), \ldots, \alpha(H_u)\}$ *and*

[5] If λ weren't coprime to the order of the point it is multiplied with, then the resulting point would have lower order, which is something $\mathcal{T}$ could detect.

– $\Phi_Q := \{[a_1]\alpha(Q_1), \dots, [a_s]\alpha(Q_s)\}$.

The inputs E, A, $\{P_1, \dots, P_r\}$ and $\{Q_1, \dots, Q_s\}$ are all honest, unprotected parameters while the sets $\{a_1, \dots, a_s\}$ and $\{H_1, \dots, H_u\}$ are secret or (honest/adversarial) protected. The outputs E_A and Φ_P are unprotected while Φ_H and Φ_Q are secret or protected. We write

$$\mathsf{ScIso}(E, A;\ \{P_1, \dots, P_r\}, \{H_1, \dots, H_u\}, \{(a_1, Q_1), \dots, (a_s, Q_s)\}) \leftarrow (E_A, \Phi_P, \Phi_H, \Phi_Q).$$

4.1 OMTUP approach

We avoid computationally intensive inversions and large scalar multiplications by presenting a delegation scheme which enables the client $\mathcal{T}$ to workonly with small multiplicative factors. We define these small factors as integers from the set $\{1, \dots, 2^t\}$, where t represents a further parameter for our delegation scheme. These small factors do not decrease the security of the scheme, since $\mathcal{T}$ further shrouds its queries with random points, but do have an impact on its verifiability.

1. **Precomputation.**
   – For each $Q_i \in E[l_j^{e_j}]$, where $j \in \{A, B\}$, $\mathcal{T}$ generates two random small integers $c_i, d_i \in \{1, \dots, 2^t\} \subset (\mathbb{Z}/l_j^{e_j}\mathbb{Z})^*$ and two random integers $f_i, g_i \in (\mathbb{Z}/l_j^{e_j}\mathbb{Z})^*$.
   – For each $H_i \in E[l_j^{e_j}]$, where $j \in \{A, B\}$, $\mathcal{T}$ generates two small integers $k_{i,1}, k_{i,2} \in \{1, \dots, 2^t\} \subset (\mathbb{Z}/l_j^{e_j}\mathbb{Z})^*$ and two random points $S_{i,1}, S_{i,2} \in E[l_j^{e_j}]$, then computes $[k_{i,1}]S_{i,1}$ and $[k_{i,2}]S_{i,2}$.
2. **Shrouding.**
   – For each a_i, $\mathcal{T}$ computes $r_i \leftarrow a_i - c_i f_i$ and $s_i \leftarrow a_i - d_i g_i$.
   – For each H_i, $\mathcal{T}$ computes $H_{1,i} \leftarrow H_i - [k_{1,i}]S_{1,i}$ and $H_{2,i} \leftarrow H_i - [k_{2,i}]S_{2,i}$.
3. **Query to $\mathcal{U}_1$.** In random order of the elements in the sets
$$\mathcal{M}_1 = \{P_1, \dots, P_r, H_{1,1}, \dots, H_{1,u}, S_{2,1}, \dots, S_{2,u}\} \text{ and}$$
$$\mathcal{N}_1 = \{(r_1, Q_1), (g_1, Q_1), \dots, (r_s, Q_s), (g_s, Q_s)\},$$
   $\mathcal{T}$ delegates
$$\mathcal{U}_1\big(E, A;\ \mathcal{M}_1, \mathcal{N}_1\big) \leftarrow (E_A, \Phi_1, \Phi_2).$$
4. **Query to $\mathcal{U}_2$.** In random order of the elements in the sets
$$\mathcal{M}_2 = \{P_1, \dots, P_r, H_{2,1}, \dots, H_{2,u}, S_{1,1}, \dots, S_{1,u}\} \text{ and}$$
$$\mathcal{N}_2 = \{(s_1, Q_1), (f_1, Q_1), \dots, (s_s, Q_s), (f_s, Q_s)\},$$
   $\mathcal{T}$ delegates
$$\mathcal{U}_2\big(E, A;\ \mathcal{M}_2, \mathcal{N}_2\big) \leftarrow (E_A, \Phi'_1, \Phi'_2).$$
5. **Verification.** Using the outputs, $\mathcal{T}$ checks,
   – if both returned curves E_A and all elements of the respective subsets $\Phi_P = \{\alpha(P_1), \dots, \alpha(P_r)\}$ are equal. If not, $\mathcal{T}$ returns $\perp$.
   – Otherwise, $\mathcal{T}$ verifies for each $i \in \{1, \dots, s\}$, if
$$[r_i]\alpha(Q_i) + [c_i]([f_i]\alpha(Q_i)) \overset{?}{=} [s_i]\alpha(Q_i) + [d_i]([g_i]\alpha(Q_i))$$
   and for each $i \in \{1, \dots, u\}$, if
$$\alpha(H_{1,i}) + [k_{1,i}]\alpha(S_{1,i}) \overset{?}{=} \alpha(H_{2,i}) + [k_{2,i}]\alpha(S_{2,i})$$
   hold. If not, $\mathcal{T}$ outputs $\perp$.
6. **Recovery.** $\mathcal{T}$ returns $(E_A, \Phi_P, \Phi_H, \Phi_Q)$, where
$$\Phi_P = \{\alpha(P_1), \dots, \alpha(P_r)\},$$
$$\Phi_H = \{\alpha(H_{1,i}) + [k_{1,i}]\alpha(S_{1,i}) \mid i = 1, \dots, u\},$$
$$\Phi_Q = \{[r_i]\alpha(Q_i) + [c_i]([f_i]\alpha(Q_i)) \mid i = 1, \dots, s\}.$$

Computational cost. Setting $u_B := |\{H_i \mid H_i \in E[l_B^{e_B}]\}|$ and using the notation introduced in Section 2 we find the following costs:

- **Delegator cost:**

$$T(h,t) = 2u\big((2h-1)A + 2Mt\big) + 2s(m + Mt) + 2u_B D.$$

- **Server cost (per server):**[6]

$$T_{\mathcal{U}} = \big(I(l_A) + S(l_A) + (r + 2u + 2s)I_P(l_A)\big)\frac{e_A}{2}\log_2 e_A + 2sMe_2.$$

- **Local computation cost:**

$$\widetilde{T} = (I(l_A) + S(l_A) + (r + u + s)I_P(l_A))\frac{e_A}{2}\log_2 e_A + sMe_2$$

- **Upload cost (per server):** $C_\uparrow = (6 + 4(r + 2u + 3s))c(p)$.
- **Download cost (per server):** $C_\downarrow = (2 + 4(r + 2u + 2s))c(p)$.

Theorem 1. *Under the OMTUP-assumption, $(T, (\mathcal{U}_1', \mathcal{U}_2'))$ constitutes a $\left(O\left(\frac{h+2.7t}{e_A \log_2 e_A}\right), 1 - \frac{l}{(l-1)2^{t+1}}\right)$-outsource-secure implementation of* ScIso*, where h and t are security parameters.*[7,8]

The proof of this theorem can be found in Appendix A.

4.2 Honest-but-curious approach

Definition 5 implies that the scalar factors a_i and the points H_i can not be shrouded using small scalars, since it strongly decreases the overall security of the delegation. Our approach is to let scalar multiplication be computed locally in order to propose an efficient delegation scheme.

1. **Precomputation.** For each $H_i \in E[l_j^{e_j}]$, where $i \in \{1, \ldots, u\}$ and $j \in \{A, B\}$, $\mathcal{T}$ runs
   - $\mathsf{GenP} \leftarrow S_i = (s_{1,i}, s_{2,i}) \in E[l_j^{e_j}]$,
   - $\mathsf{GenP}(k_i s_{1,i}, k_i s_{2,i}) \leftarrow [k_i]S_i$ for random $k_i \in (\mathbb{Z}/l_j^{e_j}\mathbb{Z})^*$.
2. **Shrouding.** For each H_i, $\mathcal{T}$ computes $H_i' \leftarrow H_i - [k_i]S_i$.
3. **Query.** In random order of the elements in the set

$$\mathcal{M} = \{P_1, \ldots, P_r, H_1', \ldots, H_u', S_1, \ldots, S_u, Q_1, \ldots, Q_s\},$$

   $\mathcal{T}$ delegates

$$\mathcal{U}\big(E, A;\ \mathcal{M}, \emptyset\big) \leftarrow (E_A, \Phi, \emptyset).$$

4. **Recovery.** Using the outputs, $\mathcal{T}$ computes

$$\begin{aligned}\Phi_H &= \{\alpha(H_i') + [k_i]\alpha(S_i) \mid i = 1, \ldots, u\}, \text{ and} \\ \Phi_Q &= \{[a_i]\alpha(Q_i) \mid i = 1, \ldots, s\},\end{aligned}$$

   and returns $(E_A, \Phi_P, \Phi_H, \Phi_Q)$ with $\Phi_P = \{\alpha(P_i) \mid i = 1, \ldots, r\}$.

[6] For simplicity, we assume $2^{e_2} \approx 3^{e_3}$, thus $S(2^{e_2}) \approx S(3^{e_3}) \approx Me_2$ via equation (2).

[7] To simplify this expression, we assumed that r,u and s are small compared to e_A.

[8] The verifiability depends on whether at least one of the parameters $\{Q_1, \ldots, Q_s\}$ or $\{H_1, \ldots, H_u\}$ is in $E[2^{e_2}]$. If that is the case, then we set $l = 2$ for the verifiability, otherwise $l = 3$. See the proof in Appendix A for more details.

Computational cost. Setting $u_B := |\{H_i \mid H_i \in E[l_B^{e_B}]\}|$, we find the following costs:

- **Delegator cost:**
$$T(h) = u\big(4hA + 2m + Me_2\big) + sMe_2 + 2u_B D.$$
- **Server cost (per server):**
$$T_{\mathcal{U}} = \big(I(l_A) + S(l_A) + (r + 2u + s)I_P(l_A)\big)\frac{e_A}{2}\log_2 e_A.$$
- **Local computation cost:**
$$\widetilde{T} = (I(l_A) + S(l_A) + (r + u + s)I_P(l_A))\frac{e_A}{2}\log_2 e_A + sMe_2$$
- **Upload cost (per server):** $C_\uparrow = (6 + 4(r + 2u + s))c(p)$.
- **Download cost (per server):** $C_\downarrow = (2 + 4(r + 2u + s))c(p)$.

Theorem 2. *Under the honest-but-curious assumption,* $(T, \mathcal{U}')$ *constitutes a* $\left(O\left(\frac{1.1h+e_2}{e_A \log_2 e_A}\right), 1\right)$*-outsource-secure implementation of* ScIso*, where* h *is a security parameter.*

The proof of this theorem can be found in Appendix A.

4.3 Special cases

The following two special cases of ScIso will be of interest in the cryptographic algorithms:

- Case $A = \mathcal{O}$: In this case, ScIso reduces to a simple scalar multiplication algorithm, without isogeny computation. The OMTUP-algorithm becomes a $\left(O\left(\frac{t}{e_A}\right), 1 - \frac{l}{(l-1)2^{t+1}}\right)$-outsource-secure implementation of ScIso, while under the honest-but-curious assumption, the algorithm becomes trivial and is not applicable anymore.
- Case $s = u = 0$: In this case, ScIso becomes an isogeny computation algorithm with only unprotected parameters. Under both the OMTUP- and the honest-but-curious assumptions, we are left with a $\left(O\left(\frac{1}{e_A \log_2 e_A}\right), 1\right)$-outsource secure implementation of ScIso.

5 Applications

5.1 Zero-knowledge proofs of identity

In this section, we show how to apply ScIso in order to decrease the computational cost of the supersingular isogeny-based zero-knowledge proof of identity (ZKPI), presented in [21] and in Section 2.2.

Peggy delegates.

1. Peggy invokes
$$\mathsf{GenP}_E \leftarrow B = (b_1, b_2)_E \in E[l_B^{e_B}] \text{ and}$$
$$\mathsf{GenP}_{E_A}(b_1, b_2) \leftarrow \alpha(B) = (b_1, b_2)_{E_A} \in E_A[l_B^{e_B}].$$
2. Then, Peggy delegates
$$\mathsf{ScIso}(E, B;\ \emptyset, \{A\}, \emptyset) \leftarrow (E_B;\ \emptyset, \{\beta(A)\}, \emptyset)$$
$$\mathsf{ScIso}(E_A, \alpha(B);\ \emptyset, \emptyset, \emptyset) \leftarrow (E_{AB};\ \emptyset, \emptyset, \emptyset)$$
3. Depending on Victor's challenge, Peggy either reveals $(B, \alpha(B))$ or $\beta(A)$.

Remark. Note that with the knowledge of B and $\beta(A)$ and assuming P_A, Q_A are public, it is possible to extract A using pairings as explained in Section 2.3. Therefore, the delegator should keep P_A, Q_A secret in the case where $\mathcal{U}$ would leak B.

Victor delegates. In Victor's case, none of the information received by Peggy is secret or (honest/adversarial) protected. Thus, Victor can simply use ScIso as follows.

1. If $b = 0$, Victor delegates

$$\mathsf{ScIso}(E, B;\ \emptyset, \emptyset, \emptyset) \leftarrow (E_B;\ \emptyset, \emptyset, \emptyset),$$
$$\mathsf{ScIso}(E_A, \alpha(B);\ \emptyset, \emptyset, \emptyset) \leftarrow (E_{AB};\ \emptyset, \emptyset, \emptyset).$$

   otherwise, if $b = 1$, he delegates

$$\mathsf{ScIso}(E_B, \beta(A);\ \emptyset, \emptyset, \emptyset) \leftarrow (E_{AB};\ \emptyset, \emptyset, \emptyset).$$

2. Victor verifies, whether the resulting elliptic curves correspond with those given by Peggy.

Remark. If Victor wants to delegate, we have to assume that Peggy's responses to Victor's challenge are honest. Otherwise, ScIso would not be outsource-secure by Theorems 1 and 2.

Cost analysis. Using the expressions from Section 4, we find the following total costs for Peggy under the OMTUP and honest-but-curious assumptions:

– **Computational cost (OMTUP):**

$$T(h, t) = 2G_{l_A}(h) + 2G_{l_B}(h) + 4Mt.$$

– **Computational cost (hbc):**

$$T(h) = 2G_{l_A}(h) + 2G_{l_B}(h) + 2m + 2A + S(l_A^{e_A}).$$

– **Local computation:**

$$\widetilde{T} = 2(S(l_B^{e_B}) + A) + (2I(l_B) + 2S(l_B) + I_P(l_B))\frac{e_B}{2}\log_2 e_B$$

– **Upload cost (both):** $C_\uparrow = 16c(p)$.
– **Download cost (both):** $C_\downarrow = 12c(p)$.

For Victor, we get the same results for both assumptions.

– **Computational cost:** $O(1)$.
– **Local computation ($b = 0$):** $\widetilde{T} = (I(l_B) + S(l_B))e_B \log_2 e_B$.
– **Local computation ($b = 1$):** $\widetilde{T} = (I(l_A) + S(l_A))\frac{e_A}{2}\log_2 e_A$.
– **Upload cost ($b = 0$):** $C_\uparrow = 12c(p)$.
– **Upload cost ($b = 1$):** $C_\uparrow = 6c(p)$.
– **Download cost ($b = 0$):** $C_\downarrow = 4c(p)$.
– **Download cost ($b = 1$):** $C_\downarrow = 2c(p)$.

Figure 2 compares the non-delegated and delegated cost for Peggy as a function of the security parameter p. We further present numerical results in the example below.

Example. Let us consider some numerical examples using the finite field characteristics $p_{503} = 2^{250}3^{159} - 1$, $p_{751} = 2^{372}3^{239} - 1$ and $p_{964} = 2^{486}3^{301} - 1$ from [1]. The table below shows the gain factor of Peggy compared to the local computation of the ZKPI-protocol, both for $h = 32$ and for truly random point generation. The first element in the parentheses is the gain factor in the OMTUP case for $t = 16$ (providing a verifiability of at least $1 - 10^{-5}$), while the second represents the honest-but-curious result. Further, we also depict the respective upload and download costs (per server) for Peggy.

	$h = 32$		truly random		$C_\uparrow$	$C_\downarrow$
	$l_A = 2$	$l_A = 3$	$l_A = 2$	$l_A = 3$		
p_{503}	(17.2,9.5)	(16.5,8.9)	(7.0,5.2)	(6.7,5.0)	1006B	755B
p_{751}	(27.6,11.6)	(26.2,10.7)	(7.8,5.6)	(7.4,5.2)	1.47kB	1.10kB
p_{964}	(36.8,12.4)	(34.9,12.2)	(8.3,5.7)	(7.8,5.5)	1.88kB	1.41kB

One can see that OMTUP is generally the more efficient implementation in terms of computational cost and that in both cases, the gain increases with the security parameter p.

5.2 Public-key encryption, SIDH and SIKE

In this section, we present how ScIso can be used in order to delegate parts of the computation of the supersingular isogeny-based public-key encryption and Diffie-Hellman key exchange protocols [21] as well as the NIST postquantum standardization candidate SIKE [33, 1]. In particular, we show how to delegate the decryption step, also known the shared key computation step within these protocols. Excluding the hash and the XOR computation in the former, both of these steps reduce to the following computation. Given a secret equivalent to $A = P_A + [a]Q_A \in E[l_A^{e_A}]$, where $\langle P_A, Q_A\rangle = E[l_A^{e_A}]$, and a public key $(E_B, \beta(P_A), \beta(Q_A))$, where $\beta : E \to E_B$, compute $E_{AB} = E_B/\langle \beta(A)\rangle$. We propose the following algorithm:

Definition 8 (The HIso-algorithm). *The* hidden isogeny algorithm HIso *takes as input a supersingular elliptic curve $E_B(\mathbb{F}_{p^2})$, with $p = l_A^{e_A} l_B^{e_B} f \mp 1$, a scalar $a \in (\mathbb{Z}/l_A^{e_A}\mathbb{Z})^*$ and two basis points $\langle \beta(P_A), \beta(Q_A)\rangle = E_B[l_A^{e_A}]$, then produces the output $E_{AB} := E_B/\langle \beta(A)\rangle$, where $\beta(A) = \beta(P_A) + [a]\beta(Q_A) \in E_B[l_A^{e_A}]$. The inputs $E_B, \beta(P_A), \beta(Q_A)$ are honest, unprotected, while a is secret or (honest/adversarial) protected. The output E_{AB} is secret or protected, thus $\langle \beta(A)\rangle$ is also secret or protected. We write*

$$\mathsf{HIso}(E_B;\ a, \beta(P_A), \beta(Q_A)) \leftarrow E_{AB}.$$

We implement HIso using ScIso as a subroutine. We also introduce a new security parameter k.

1. **Precomputation.** From its secret a, $\mathcal{T}$ computes al_A^k.
2. **Shrouding.** $\mathcal{T}$ invokes
$$\mathsf{ScIso}(E_B, \mathcal{O};\ \emptyset, \emptyset, \{(l_A^k, \beta(P_A)), (al_A^k, \beta(Q_A))\} \\ \leftarrow (E_B, \emptyset, \emptyset, \{[l_A^k]\beta(P_A), [al_A^k]\beta(Q_A)\})$$
and computes $\beta(\tilde{A}) = [l_A^k]\beta(P_A) + [al_A^k]\beta(Q_A)$, so that $\langle \beta(\tilde{A})\rangle = \langle [l_A^k]\beta(A)\rangle$.
3. **Query.** $\mathcal{T}$ delegates the isogeny computations using
$$\mathsf{ScIso}(E_B, \beta(\tilde{A});\ \{\beta(P_A)\}, \emptyset, \{(a, \beta(Q_A))\} \\ \leftarrow (E_{\tilde{A}B}, \{\tilde{\alpha}(\beta(P_A))\}, \emptyset, \{[a]\tilde{\alpha}(\beta(Q_A))\})$$
where $\tilde{\alpha} : E_B \to E_{\tilde{A}B} := E_B/\langle \beta(\tilde{A})\rangle$ is an $l_A^{e_A - k}$-isogeny, since $|\langle [l_A^k]A\rangle| = l_A^{e_A-k}$.

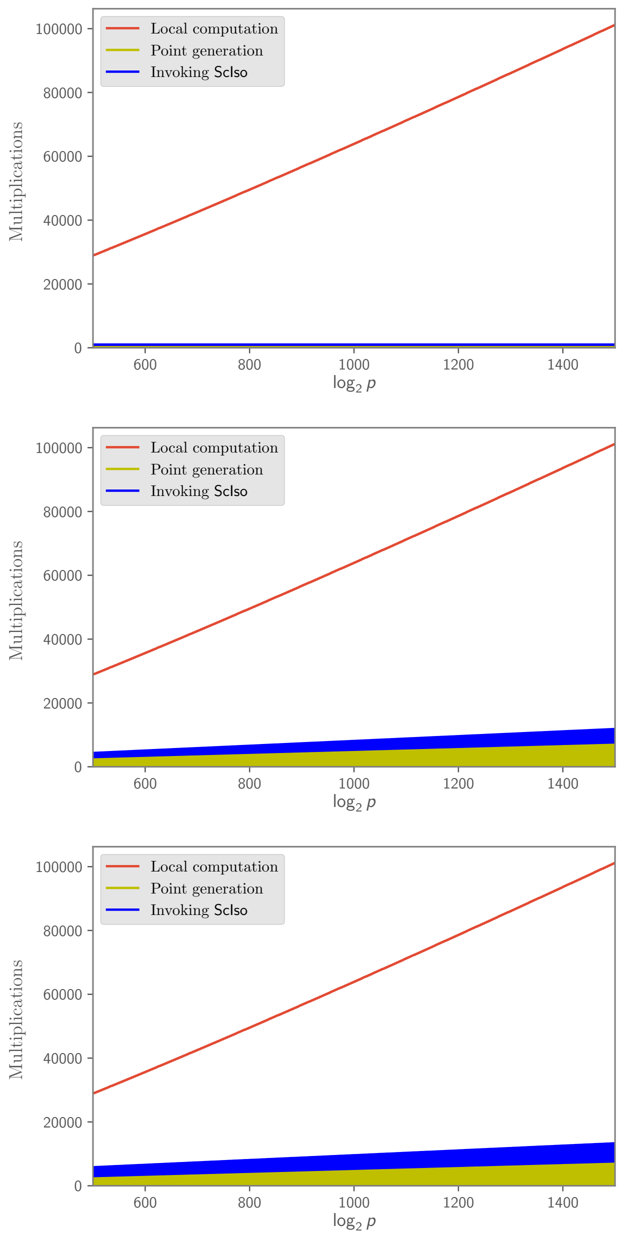


Fig. 2. Theoretical comparison of the local, non-delegated cost of the ZKPI protocol (in red) versus the delegated cost for the prover, Peggy, assuming $l_A = 3$. The delegator's cost is split up into the generation of B and $\alpha(B)$ (in yellow) and the invocation of ScIso (in blue). The top graph shows the OMTUP-case where $h = 32$, while the middle graph shows OMTUP with truly random point generation. In both cases, we chose $t = 16$. The bottom graph shows the honest-but-curious case with truly random point generation. The costs are expressed in multiplications over the underlying finite field $\mathbb{F}_{p^2}$.

4. **Verification.** If either of the outputs of steps 2 or 3 is $\perp$, then $\mathcal{T}$ outputs $\perp$.
5. **Recovery.** From the outputs, $\mathcal{T}$ computes the kernel generator $\tilde{\alpha}(\beta(P_A)) + [a]\tilde{\alpha}(\beta(Q_A))$, which generates $\langle \tilde{\alpha}(\beta(A)) \rangle$, then computes the l_A^k-isogeny
$$\bar{\alpha} : E_{\tilde{A}B} \to E_{AB} \quad \text{with} \quad \ker \bar{\alpha} = \langle \tilde{\alpha}(\beta(A)) \rangle,$$
itself. Then T returns $E/\langle A \rangle$.

Remark. Note that $[l_A^k]A = (l_A^k a_1, l_A^k a_2)$ does not disclose $\langle A \rangle$ by using pairings as described in Section 2.3, since l_A^k is not invertible in $\mathbb{Z}/l_A^{e_A}\mathbb{Z}$. A potential attack option would be to consider the underlying isogeny graph, i.e. let $\tilde{\alpha}_1 : E_B \to E_{\tilde{A}B}$ be a $l_A^{e_A - k}$-isogeny. From $E_{\tilde{A}B}$, there are now exactly l_A^k possible paths of length k to take (without backtracking) leading to l_A^k distinct elliptic curves, however only one of which is the hidden E_{AB}. Even if the adversary could identify E_{AB} (e.g. by being able to decrypt an intercepted ciphertext), then it is still unfeasible to generate all possible curves, if k is chosen to be large enough. In particular, we can consider this problem as a database search, so that Grover's algorithm [18] is applicable for a quantum adversary. We propose to choose k in order to reflect the security defined by the underlying finite field characteristic p. This implies that $k \geq \frac{1}{3} \log_l p$. In that sense, k defines the security of the shrouding step.

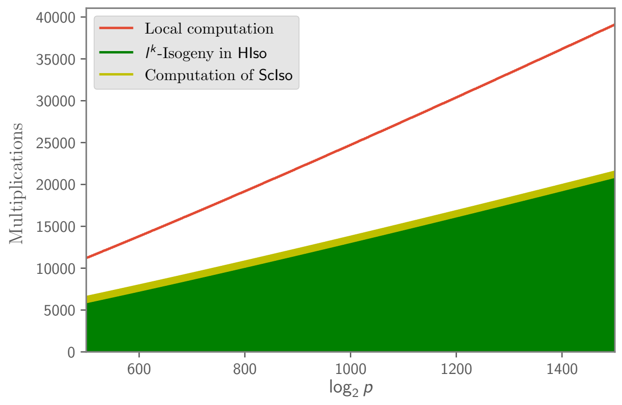
Cost analysis. Using the expressions from Section 4, we find the following total costs under the OMTUP and honest-but-curious assumptions:

– **Computational cost (OMTUP):**
$$T(t,k) = 7m + 2A + 6Mt + (I(l_A) + S(l_A))\frac{k}{2}\log_2 k.$$
– **Computational cost (hbc):**
$$T(k) = m + 2A + 3S(l_A^{e_A}) + (I(l_A) + S(l_A))\frac{k}{2}\log_2 k.$$
– **Local computation cost:**
$$\widetilde{T} = S(l_A^{e_A}) + A + (I(l_A) + S(l_A))\frac{e_A}{2}\log_2(e_A).$$
– **Upload cost (OMTUP):** $C_\uparrow = 48c(p)$.
– **Download cost (OMTUP):** $C_\downarrow = 32c(p)$.
– **Upload cost (hbc):** $C_\uparrow = 28c(p)$.
– **Download cost (hbc):** $C_\downarrow = 20c(p)$.

We note that the honest-but-curious case has a high computational cost for the delegator, and it actually turns out, that this cost is very close to the local computation of the decryption step, so that our proposed algorithm is no longer recommendable under the honest-but curious assumption. Nevertheless, we still do obtain a non-negligible gain using the OMTUP assumption. Figure 3 compares the non-delegated and delegated cost of the decryption step as a function of the security parameter p. We further present numerical results in the example below.

Example. The table below shows the gain factor of the delegation compared to the local computation of the decryption step. Again, we also depict the respective upload and download costs (per server) for the delegator.

	$l_A = 2$	$l_A = 3$	$C_\uparrow$	$C_\downarrow$
p_{503}	1.7	1.7	2.9kB	2.0kB
p_{751}	1.8	1.8	4.4kB	2.9kB
p_{964}	1.8	1.8	5.6kB	3.8kB



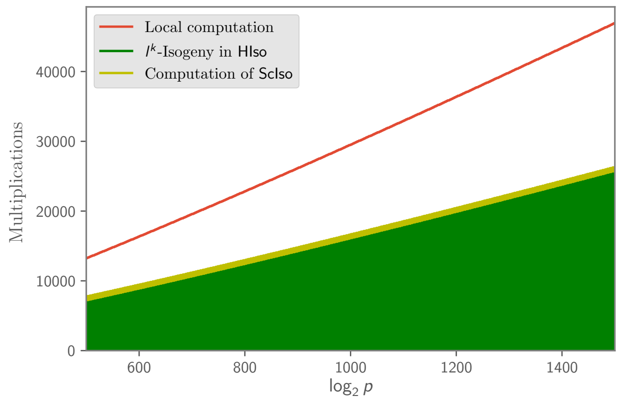


Fig. 3. Theoretical comparison of the local, non-delegated cost of the decryption step (in red) versus the delegated cost for $\mathcal{T}$ under the OMTUP assumption. The delegator's cost is split up into the invocations of ScIso (yellow) and the recovery step, i.e. the computation of the l_A^k-isogeny (green). The top graph shows the case $l_A = 2$, while the bottom graph shows $l_A = 3$. In both cases, we chose $t = 16$ and set $k = \frac{1}{3}\log_{l_A} p$. The costs are expressed in multiplications over the underlying finite field $\mathbb{F}_{p^2}$.

Communication complexity. In our proposed scheme, the delegator has to invoke ScIso twice leading to an elevated communication cost. Nevertheless, we note for example that tiny sensor nodes like MICAz and TelosB require much more energy even for a single scalar multiplication than a single round of communication [12]. Since isogeny computations require much more energy than scalar multiplications, it seems to be still highly worthwhile to delegate the isogeny computation. Especially in application scenarios, in which communication and bandwidth do not dominate the overall costs, our schemes could be effectively deployed to speed up computational costs.

6 Benchmarks

In this section, we present benchmarks of the delegated versus local computation of the cryptographic schemes presented in Section 5 in order to support our theoretical predictions. Figure 4 compares the delegated and the local cost of the ZKPI protocol, while Figure 5 does the same for the key decryption steps. For these results, we used the PQCrypto-SIDH v3.0 library, developed by Microsoft Research [23], which implements SIDH on the two finite fields $\mathbb{F}_{p^2}$ with $p_{503} = 2^{250} \cdot 3^{159} - 1$ and $p_{751} = 2^{372} \cdot 3^{239} - 1$. Our benchmarks were done on a 2.8GHz Intel i5-8400 running Ubuntu 18.04 LTS.

In the ZKPI-case, we can see that the implemented results present the same behavior as our theoretical predictions. The gain factors in the OMTUP-case are even slightly higher than predicted in the example in Section 5.1. These results are presented in the table below, where the first entry in each parenthesis shows the OMTUP-case while the second shows the honest-but-curious case.

	$h = 32$		truly random	
	$l_A = 2$	$l_A = 3$	$l_A = 2$	$l_A = 3$
p503	(21.9,9.2)	(19.7,8.3)	(10.0,6.1)	(9.0,5.5)
p751	(34.1,10.5)	(30.1,9.3)	(10.3,6.2)	(9.1,5.5)

We also see that the relative performance of our ZKPI-delegation scheme does improve with higher security levels.

In the decryption-case, however, we find slightly lower gain factors than predicted e.g. for $l = 3$, we find 1.42 and 1.47 for $p503$ and $p751$, respectively. We assume that this has to do with the efficient implementation of the local computation in the PQCrypto-SIDH v3.0 library, from which our implementation doesn't benefit (yet).

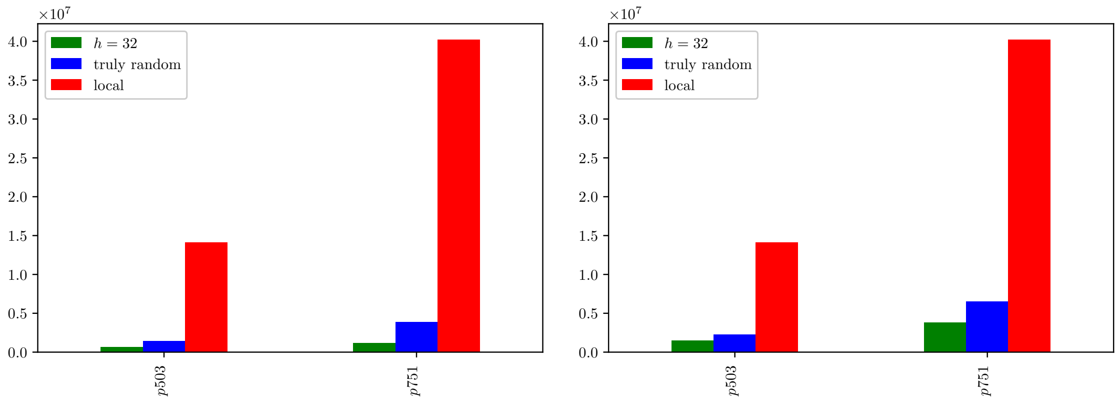


Fig. 4. Comparison of the local, non-delegated cost of the decryption step (red) versus the delegated cost for $\mathcal{T}$ for $h = 32$ (green) and for truly random point generation (blue) assuming $l_A = 3$. The left graph shows the OMTUP case for $t = 16$, while the right graph shows the honest-but-curious assumption. The costs are expressed in CPU cycles.

7 Conclusion

In this paper, we initiated first steps of study towards making post-quantum cryptographic primitives efficiently deployable for resource-constrained environments by securely delegating supersingular elliptic curve isogeny computations to external servers with reasonable verifiability guarantees. We propose a secure delegation algorithm, ScIso, implemented using one honest server, that may try to extract useful information (honest-but-curious assumption) or using two servers, only one of which is assumed to be malicious (OMTUP-assumption).
Using our algorithms as subroutines in isogeny-based zero-knowledge proofs of identity, we effectively reduce the computational cost for the proving party to about 3% of the original cost at the

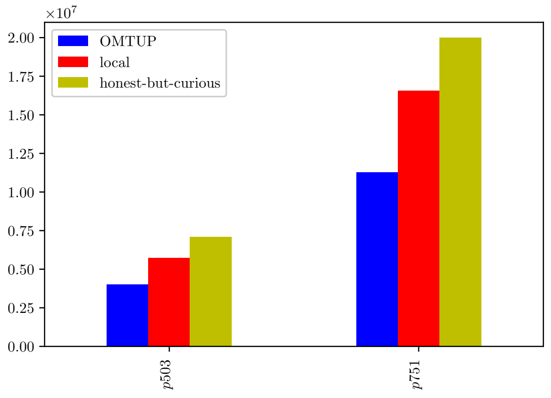


Fig. 5. Comparison of the local, non-delegated cost of the decryption step (red) versus the delegated cost for $\mathcal{T}$ under the OMTUP assumption (blue) and, for the sake of comparison, under the honest-but-curious assumption (yellow), assuming $l_A = 3$. The costs are expressed in CPU cycles.

128−bit quantum security level, while the cost of the verifying party simply reduces to communication costs with the servers and comparison operations. We also show, that this relative cost further decreases with higher security. Even when repeating the protocol for multiple rounds (e.g. 10−20 rounds), the delegated ZKPI-protocol is computationally less expensive than a single round in the non-delegated SIDH or SIKE protocols, at least for the delegator. Both our efficiency analysis and experimental results show the practical deployability of our algorithms.

As a second application, we use ScIso as a subroutine of another delegation algorithm called HIso, which computes a secret elliptic curve from a public key and can therefore be used at the decryption step of supersingular isogeny-based public key encryption (PKE) or key encapsulation (SIKE), as well as in the shared key computation step within SIDH. The cost of this step is more or less reduced to 60% of the original cost at the 128−bit quantum security level.

As a next step it would suggest itself to try increase the gain of this decryption step, i.e. to find a more efficient way of hiding the kernel and the codomain while delegating an isogeny computation, under both the OMTUP- and honest-but-curious assumptions. It would be of interest to find delegation algorithms for the public-key generation step in SIDH or SIKE, in order to make these protocols efficiently deployable on resource-constrained devices. Furthermore, one could explore similar algorithms for other isogeny-based protocols, such as verifiable delay functions or [16] or cryptographic hash functions [14] based on supersingular elliptic curve isogenies. We note that delegating the supersingular isogeny-based hash computation from [14] is trivial under the OMTUP- and honest-but-curious assumptions if the message m is public, but not for a secret message.

Another open problem would be to extend our results to delegation assumptions without a honest party, such as the two-untrusted-program model (TUP-assumption) and the one-untrusted-program model (OUP-assumption) from [19] with reasonable verifiability properties, or to otherwise prove their impossibilities.

Finally, it suggests itself to find secure delegation schemes for other post-quantum cryptographic primitives, such as those used in lattice-based or code-based cryptography.

References

1. Azarderakhsh, R., Campagna, M., Costello, C., De Feo, L., Hess, B., Jalali, A., Jao, D., Koziel, B., LaMacchia, B., Longa, P., et al.: Supersingular isogeny key encapsulation. Submission to the NIST Post-Quantum Standardization project (2017)
2. Bernstein, D., Lange, T.: Explicit-formulas database. https://www.hyperelliptic.org/EFD (2019)

3. Boyko, V., Peinado, M., Venkatesan, R.: Speeding up discrete log and factoring based schemes via precomputations. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 221–235. Springer (1998)
4. Charles, D.X., Lauter, K.E., Goren, E.Z.: Cryptographic hash functions from expander graphs. Journal of Cryptology pp. 93–113 (2009)
5. Chen, X., Li, J., Ma, J., Tang, Q., Lou, W.: New algorithms for secure outsourcing of modular exponentiations. IEEE Transactions on Parallel and Distributed Systems **25**(9), 2386–2396 (2014)
6. 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)
7. Childs, A.M., Jao, D., Soukharev, V.: Constructing elliptic curve isogenies in quantum subexponential time. J. Mathematical Cryptology **8**(1), 1–29 (2014)
8. Costello, C., Hisil, H.: A simple and compact algorithm for sidh with arbitrary degree isogenies. In: Advances in Cryptology – ASIACRYPT 2017. pp. 303–329 (2017)
9. Costello, C., Hisil, H.: A simple and compact algorithm for sidh with arbitrary degree isogenies. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 303–329. Springer (2017)
10. Costello, C., Smith, B.: Montgomery curves and their arithmetic: The case of large characteristic fields. IACR Cryptology ePrint Archive **2017**, 212 (2017)
11. Couveignes, J.: Hard homogeneous spaces (2006), https://eprint.iacr.org/2006/291.pdf
12. De Meulenaer, G., Gosset, F., Standaert, F.X., Pereira, O.: On the energy cost of communication and cryptography in wireless sensor networks. In: 2008 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications. pp. 580–585. IEEE (2008)
13. De Rooij, P.: Efficient exponentiation using precomputation and vector addition chains. In: Workshop on the Theory and Application of of Cryptographic Techniques. pp. 389–399. Springer (1994)
14. Doliskani, J., Pereira, G.C., Barreto, P.S.: Faster cryptographic hash function from supersingular isogeny graphs. IACR Cryptology ePrint Archive **2017**, 1202 (2017)
15. Faz-Hernández, A., López, J., Ochoa-Jiménez, E., Rodríguez-Henríquez, F.: A faster software implementation of the supersingular isogeny diffie-hellman key exchange protocol. IEEE Transactions on Computers **67**(11), 1622–1636 (2018)
16. Feo, L.D., Masson, S., Petit, C., Sanso, A.: Verifiable delay functions from supersingular isogenies and pairings (2019), https://eprint.iacr.org/2019/166.pdf
17. Galbraith, S.D., Petit, C., Shani, B., Ti, Y.B.: On the security of supersingular isogeny cryptosystems. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 63–91. Springer (2016)
18. Grover, L.K.: A fast quantum mechanical algorithm for database search. arXiv preprint quant-ph/9605043 (1996)
19. Hohenberger, S., Lysyanskaya, A.: How to securely outsource cryptographic computations. In: Theory of Cryptography Conference. pp. 264–282. Springer (2005)
20. Jalali, A., Azarderakhsh, R., Kermani, M.M., Jao, D.: Towards optimized and constant-time csidh on embedded devices (2019), https://eprint.iacr.org/2019/297.pdf
21. 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)
22. Kiraz, M.S., Uzunkol, O.: Efficient and verifiable algorithms for secure outsourcing of cryptographic computations. International Journal of Information Security **15**(5), 519–537 (2016)
23. Microsoft Research: Pqcrypto-sidh v3.0 library. https://github.com/Microsoft/PQCrypto-SIDH (2019)
24. Montgomery, P.L.: Speeding the pollard and elliptic curve methods of factorization. Mathematics of computation **48**(177), 243–264 (1987)
25. Moritz, S., Uzunkol, O.: A more efficient secure fully verifiable delegation scheme for simultaneous group exponentiations. In: International Conference on Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments. pp. 74–93. Springer (2018)
26. Nguyen, P.Q., Shparlinski, I.E., Stern, J.: Distribution of modular sums and the security of the server aided exponentiation. In: Cryptography and Computational Number Theory, pp. 331–342. Springer (2001)
27. NIST: Nist reveals 26 algorithms advancing to the post-quantum crypto 'semifinals' (2019), https://www.nist.gov/news-events/news/2019/01/nist-reveals-26-algorithms-advancing-post-quantum-crypto-semifinals

28. Petit, C.: Faster algorithms for isogeny problems using torsion point images. In: Advances in Cryptology – ASIACRYPT 2017. pp. 330–353 (2017)
29. Renes, J.: Computing isogenies between montgomery curves using the action of (0, 0). In: International Conference on Post-Quantum Cryptography. pp. 229–247. Springer (2018)
30. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Conference on the Theory and Application of Cryptology. pp. 239–252. Springer (1989)
31. Schnorr, C.P.: Efficient signature generation by smart cards. Journal of cryptology **4**(3), 161–174 (1991)
32. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: Proceedings of the 35th Annual Symposium on Foundations of Computer Science. pp. 124–134 (1994)
33. SIKE: Supersingular isogeny key encapsulation (2018), https://sike.org
34. Silverman, J.H.: The arithmetic of elliptic curves, vol. 106. Springer Science & Business Media (2009)
35. Sommerseth, M.L., Hoeiland, H.: Pohlig-hellman applied in elliptic curve cryptography (2015)
36. Stolbunov, A.: Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves. Adv. in Math. of Comm. **4**(2), 215–235 (2010)
37. 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)
38. Wang, Y., Wu, Q., Wong, D.S., Qin, B., Chow, S.S., Liu, Z., Tan, X.: Securely outsourcing exponentiations with single untrusted program for cloud storage. In: European Symposium on Research in Computer Security. pp. 326–343. Springer (2014)
39. Winkler, N.: The discrete log problem and elliptic curve cryptography
40. Yoo, Y., Azarderakhsh, R., Jalali, A., Jao, D., Soukharev, V.: A post-quantum digital signature scheme based on supersingular isogenies. In: Financial Cryptography and Data Security. pp. 163–181. Cham (2017)

A Security proofs

A.1 Proof of Theorem 1

Correctness. Because of the homomorphism property of isogenies, it holds that

$$\begin{aligned} [r_i]\alpha(Q_i) + [c_i]([f_i]\alpha(Q_i)) &= [a_i]\alpha(Q_i), \quad \text{for } i = 1, \dots, s \quad \text{and} \\ \alpha(H_{1,i}) + [k_{1,i}]\alpha(S_{1,i}) &= \alpha(H_i), \qquad \text{for } i = 1, \dots, u. \end{aligned}$$

Correctness is trivial for Φ_P.

Verifiability. Any of the two servers could only cheat if they knew any of the small parameters used in the shrouding step. Let for example $c, d \in \{1, \dots, 2^t\} \subset (\mathbb{Z}/l^e\mathbb{Z})^*$ and $f, g \in \mathbb{Z}/l^e\mathbb{Z}$ be parameters in order to shroud a scalar factor $a \in \{a_i \mid i = 1, \dots, s\} \subset (\mathbb{Z}/l^e\mathbb{Z})^*$ of the public point $Q \in E[l^e]$ and let similarly $k_1, k_2 \in \{1, \dots, 2^t\} \subset (\mathbb{Z}/l^e\mathbb{Z})^*$ denote parameters used to shroud a point $H \in E[l^e]$ as explained in the first two steps of the algorithm in Section 4.1. Let us assume $\mathcal{U}_1'$ were the cheater and knew the security parameter t. Then, for H, $\mathcal{U}_1'$ could potentially choose a point $X \in E[l^e]$ and return the following, wrong results

$$H_1 \mapsto \alpha(H_1 + [k_2]X), S_2 \mapsto \alpha(S_2 + X),$$

or, similarly return

$$(r, Q) \mapsto [r]Q + [d]X, (g, Q) \mapsto [g]Q + X,$$

both of which would satisfy their respective verification condition, but return wrong results to $\mathcal{T}$. However, this is only possible, if $\mathcal{U}_1'$ can somehow distinguish all queries and if it knew k_2 or d, respectively. Any other combination would not satisfy the verification conditions and T would realize the deceit. Since both parameters come from the set $\{1, \dots, 2^t\} \subset (\mathbb{Z}/l^e\mathbb{Z})^*$ of order $\frac{l-1}{l}2^t$, and assuming both queries are indistinguishable to $\mathcal{U}_1'$, the probability for $\mathcal{U}_1'$ to correctly guess the respective factor is $\frac{l}{(l-1)2^{t+1}}$. If there are more queries by $\mathcal{T}$, or if $\mathcal{U}_1'$ tries to cheat on multiple queries, this probability further decreases. We therefore assume that $\mathcal{U}_1'$ tries to cheat on a single point in order to maximize its chance for success. Given these considerations, the odds of $\mathcal{T}$ detecting a wrong result by $\mathcal{U}_1'$ is given by at least $\beta_l(t) = 1 - \frac{l}{(l-1)2^{t+1}}$. Note that $\beta_3(t) > \beta_2(t)$, so that $\beta_2(t)$ presents the lower bound. A server trying to cheat would ideally always try to cheat on a 2^{e_2}-torsion element in order to increase its probability for success. Thus, if there is at least one 2^{e_2}-torsion element within $\{Q_1, \ldots, Q_s\} \cup \{H_1, \ldots, H_u\}$, then we assume the verifiability to be $\beta_2(t)$, otherwise $\beta_3(t)$. For the sake of generality, we will denote the general verifiability as $\beta_l(t)$ in the theorems.

Security. Let $\mathcal{A} = (\mathcal{E}, \mathcal{U}_1', \mathcal{U}_2')$ be a PPT adversary that interacts with a PPT algorithm $\mathcal{T}$ in the OMTUP model. Since the procedure is exactly the same for each point (or pair) in the three input sets, we will reduce our analysis to a single point per set. Furthermore, we will analyze the security of the three sets individually for the sake of overview. Note, that the security of E, A and the sets $\{P_1, \ldots, P_r\}$ and $\{Q_1, \ldots, Q_s\}$ are trivially given, since all of these parameters are unprotected. We therefore reduce the rest of this analysis to points in the sets $\{H_1, \ldots, H_u\}$ and $\{a_1, \ldots, a_s\}$.

Pair One: $\mathcal{E}VIEW_{\text{real}} \sim \mathcal{E}VIEW_{\text{ideal}}$
Let $H \in E[l^e]$ be a point in $\{H_1, \ldots, H_u\}$. If the input H is anything but secret, the simulation is trivial and the PPT simulator $\mathcal{S}_1$ behaves exactly as in the real execution. If H is secret, then on receiving the input on round i, $\mathcal{S}_1$ ignores it and generates four random points $X_1, \ldots, X_4 \in E[l^e]$ and makes the queries

$$\begin{aligned}\mathcal{U}_1(E, A;\ \{X_1, X_2\}, \{\ldots\}) &\leftarrow (E_A, \{\alpha(X_1), \alpha(X_2)\}, \{\ldots\}),\\ \mathcal{U}_2(E, A;\ \{X_3, X_4\}, \{\ldots\}) &\leftarrow (E_A, \{\alpha(X_3), \alpha(X_4)\}, \{\ldots\}),\end{aligned}$$

to the servers, then verifies if all the outputs are correct. If they are, $\mathcal{S}_1$ returns $(Y_p^i, Y_u^i, replace^i) = (\emptyset, \emptyset, 0)$. Otherwise, if the curves E_A do not match, $\mathcal{S}_1$ returns $(Y_p^i, Y_u^i, replace^i) = (\text{"}error\text{"}, \emptyset, 1)$, else $\mathcal{S}_1$ generates a random $\xi \in \{1, \ldots, (1 - l_B^{-1})2^{t+1}\}$. If $\xi \neq 1$, it returns $(Y_p^i, Y_u^i, replace^i) = (\text{"}error\text{"}, \emptyset, 1)$, otherwise it generates a random $Y \in E_A[l^e]$ and returns $(Y_p^i, Y_u^i, replace^i) = (Y, \emptyset, 1)$. In either case, $\mathcal{S}_1$ saves the appropriate states.

The input distributions to $(\mathcal{U}_1', \mathcal{U}_2')$ in the real and ideal experiments are computationally indistinguishable. In the real scenario, we generate two random points using our lookup-tables, which are either truly random or computationally indistinguishable from random, while $\mathcal{S}_1$ chooses the inputs uniformly at random. We now have four scenarios to consider. (1) If $(\mathcal{U}_1', \mathcal{U}_2')$ behave honestly in the ith round, then $\mathcal{E}VIEW^i_{\text{real}} \sim \mathcal{E}VIEW^i_{\text{ideal}}$, because in the real experiment $T^{(\mathcal{U}_1', \mathcal{U}_2')}$ perfectly executes ScIso and in the ideal experiment $\mathcal{S}_1$ chooses not to replace the output of ScIso. If however one of $(\mathcal{U}_1', \mathcal{U}_2')$ returns a wrong results in the ith round, then it will be caught by both T and $\mathcal{S}_1$ with probability 1 or $1 - \frac{l_B}{(l_B-1)2^{t+1}}$, in the case of the (2) wrong elliptic curve or (3) wrong point image, respectively, resulting in an output of "error". (4) Otherwise, the servers would actually succeed in returning an undetected wrong output. Since in the real scenario, the output would look wrong, but random to $\mathcal{E}$, $\mathcal{S}_1$ returns a random l^e-torsion point on the codomain curve.

Similar arguments hold for a scalar factor $a \in \mathbb{Z}/l^e\mathbb{Z}$ from $\{a_1, \ldots, a_s\}$, except that in each round, $\mathcal{S}_1$ produces four random factors $x_1, \ldots, x_4 \in \mathbb{Z}/l^e\mathbb{Z}$ as queries, then proceeds analogously to the case with the point H. We assume the factors to be randomly generated by $\mathcal{T}$, and are thus indistinguishable from the real experiment.

In both cases, even when one of $(\mathcal{U}_1', \mathcal{U}_2')$ behaves dishonestly in the ith round, we have $\mathcal{E}VIEW^i_{\text{real}} \sim \mathcal{E}VIEW^i_{\text{ideal}}$. By the hybrid argument, we can then conclude, that $\mathcal{E}VIEW_{\text{real}} \sim \mathcal{E}VIEW_{\text{ideal}}$.

Pair Two: $\mathcal{U}VIEW_{\text{real}} \sim \mathcal{U}VIEW_{\text{ideal}}$
Whether H is secret or (honest/adversarial) protected, the same PPT simulator $\mathcal{S}_2$ will work. On receiving the input on round i, $\mathcal{S}_2$ ignores it and generates four random points $X_1, \dots, X_4 \in E[l^e]$ and makes the following queries to the servers:

$$\mathcal{U}_1(E, A;\ \{X_1, X_2\}, \{\dots\}) \leftarrow (E_A, \{\alpha(X_1), \alpha(X_2)\}, \{\dots\}),$$
$$\mathcal{U}_2(E, A;\ \{X_3, X_4\}, \{\dots\}) \leftarrow (E_A, \{\alpha(X_3), \alpha(X_4)\}, \{\dots\}),$$

Then $\mathcal{S}_2$ saves its own state and the states of $(\mathcal{U}_1', \mathcal{U}_2')$. Similarly, for a, $\mathcal{S}_2$ produces four random factors $x_1, \dots, x_4 \in \mathbb{Z}/l^e\mathbb{Z}$ as queries, which are indistinguishable from the real experiment. $\mathcal{E}$ can easily distinguish between the real and ideal experiments, but it can not communicate that information to $(\mathcal{U}_1', \mathcal{U}_2')$, since in the ith round, T always re-randomizes his inputs to $(\mathcal{U}_1', \mathcal{U}_2')$ and in the ideal experiment, $\mathcal{S}_2$ always creates new, random queries for $(\mathcal{U}_1', \mathcal{U}_2')$.

Thus, for each round, we have $\mathcal{U}VIEW^i_{\text{real}} \sim \mathcal{U}VIEW^i_{\text{ideal}}$, which, by the hybrid argument yields $\mathcal{U}VIEW_{\text{real}} \sim \mathcal{U}VIEW_{\text{ideal}}$.

Efficiency. The efficiency gain can be computed as the fraction of the delegator cost and the local computation cost as follows.

$$\alpha(h, t) = \frac{T(h, t)}{\widetilde{T}} = \frac{2u\big((2h-1)A + 2Mt\big) + 2s(m + Mt) + 2u_B D}{(I(l_A) + S(l_A) + (r + u + s)I_P(l_A))\frac{e_A}{2}\log_2 e_A + sMe_2}$$

Assuming the factors r, u, s are small and of similar value (in cryptosystems, generally $r, u, s \in \{0, 1, 2\}$), we can simplify this expression to

$$\alpha(h, t) = O\left(\frac{4hA + 6Mt}{e_A \log_2 e_A}\right) = O\left(\frac{h + 2.7t}{e_A \log_2 e_A}\right).$$

$\square$

A.2 Proof sketch of Theorem 2

The proof of Theorem 2 is very similar to the proof of Theorem 1. Rather than give the full proof here, we will simply sketch it and point out the major differences to the OMTUP case. Note that the verifiability is always equal to 1 in the honest-but-curious case.

Correctness. Because of the homomorphism property of isogenies, it holds that

$$\alpha(H_i') + [k_i]\alpha(S_i) = \alpha(H_i),\ \text{for } i = 1, \dots, u.$$

Correctness is trivial for Φ_P and Φ_Q.

Security. Let $\mathcal{A} = (\mathcal{E}, \mathcal{U})$ be a PPT adversary that interacts with a PPT algorithm $\mathcal{T}$ in the honest-but-curious model. As in the OMTUP case the security of E, A and the sets $\{P_1, \dots, P_r\}$ and $\{Q_1, \dots, Q_s\}$ are trivially given, since all of these parameters are unprotected. Similarly the security of the secret or protected parameters $\{a_1, \dots, a_s\}$ are given, since they are never disclosed to either $\mathcal{U}$ or $\mathcal{E}$. Our analysis therefore reduces to points in the sets $\{H_1, \dots, H_u\}$.

Pair One: $\mathcal{E}VIEW_{\text{real}} \sim \mathcal{E}VIEW_{\text{ideal}}$
This case works completely analogous to the proof in the OMTUP-case, except that $\mathcal{S}_1$ generates two random points $X_1, X_2 \in E[l^e]$ and queries

$$\mathcal{U}(E, A;\ \{X_1, X_2\}, \{\dots\}) \leftarrow (E_A, \{\alpha(X_1), \alpha(X_2)\}, \{\dots\}),$$

then outputs $(Y_p^i, Y_u^i, replace^i) = (\emptyset, \emptyset, 0)$ and saves the appropriate states. As noted in the OMTUP-case, the input distributions to $\mathcal{U}'$ in the real and ideal experiments are computationally indistinguishable. Since we assume $\mathcal{U}'$ to behave honestly, it perfectly executes ScIso in the ith round and by the hybrid argument, we have, that $\mathcal{E}VIEW_{\text{real}} \sim \mathcal{E}VIEW_{\text{ideal}}$.

Pair Two: $\mathcal{U}VIEW_{\text{real}} \sim \mathcal{U}VIEW_{\text{ideal}}$
This case also works completely analogous to the proof in the OMTUP-case, except that $\mathcal{S}_2$ generates two random points $X_1, X_2 \in E[l^e]$ and queries

$$\mathcal{U}(E, A;\ \{X_1, X_2\}, \{\dots\}) \leftarrow (E_A, \{\alpha(X_1), \alpha(X_2)\}, \{\dots\}),$$

then saves its own state and the state of $\mathcal{U}'$. Since the input distributions to $\mathcal{U}'$ are computationally indistinguishable in the real and ideal experiments and $\mathcal{U}'$, the hybrid argument yields $\mathcal{U}VIEW_{\text{real}} \sim \mathcal{U}VIEW_{\text{ideal}}$.

Efficiency. We find

$$\alpha(h,t) = \frac{T(h)}{\widetilde{T}} = \frac{u\big(4hA + 2m + Me_2\big) + sMe_2 + 2u_B D}{(I(l_A) + S(l_A) + (r+u+s)I_P(l_A))\frac{e_A}{2}\log_2 e_A + sMe_2}$$

Again, assuming small r, u, s, we simplify this expression to

$$\alpha(h) = O\left(\frac{4hA + 2Me_2}{e_A \log_2 e_A}\right) = O\left(\frac{1.1h + e_2}{e_A \log_2 e_A}\right).$$

□