Delegating Supersingular Isogenies over $\mathbb{F}_{p^2}$ with Cryptographic Applications

Robi Pedersen[1] and Osmanbey Uzunkol[2]

robi.pedersen@esat.kuleuven.be, osmanbey.uzunkol@gmail.com,

[1] imec-COSIC, ESAT, KU Leuven, Belgium
[2] Information und Kommunikation, Flensburg University of Applied Sciences, Flensburg, Germany

Abstract. Although isogeny-based cryptographic schemes enjoy the lowest key sizes amongst current post-quantum cryptographic candidates, they unfortunately come at a high computational cost, which makes their deployment on the ever-growing number of resource-constrained devices difficult. Speeding up the expensive post-quantum cryptographic operations by delegating these computations from a weaker client to untrusted powerful external servers is a promising approach. Following this, we present in this work mechanisms allowing computationally restricted devices to securely and verifiably delegate isogeny computations to potentially untrusted third parties. In particular, we propose two algorithms that can be seamlessly integrated into existing isogeny-based protocols and which lead to a much lower cost for the delegator than the full, local computation. For example, compared to the local computation cost, we reduce the public-key computation step of SIDH/SIKE [30,39] by a factor 5 and the zero-knowledge proof of identity from [30] by a factor 16 for the prover, while it becomes almost free for the verifier, respectively, at the NIST security level 1.

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

1 Introduction

Delegation of Cryptographic Primitives. In recent years, interconnected devices using new computational paradigms such as cloud, edge and mobile computing, and interactions of those with technologies like the industrial internet of things, big data and artificial intelligence, are steadily increasing in numbers. As a result, delegating expensive computations from clients such as RFID-cards and low power sensors with constrained resources or capabilities to powerful external resources has become a highly active and an indispensable research and development area. In particular, delegation of costly computations to much more powerful external devices with large-scale computational capabilities has gained an increasing interest, not only among researchers, but especially also among

practitioners requiring to utilize these technologies effectively within their (potential) solutions/products while guaranteeing sufficient interoperability.

Delegation of sensitive computation to *potentially malicious* external devices and services, however, comes with some additional challenges, such as requiring security of the clients' inputs/outputs as well as verifiability of the outputs coming from these external devices and services. A particular case of interest is the delegation of cryptographic algorithms and protocols. The security and verifiability properties of cryptographic delegations were first formalized in a security model introduced by Hohenberger and Lysyanskaya [29], introduced in the context of the delegation of modular exponentiations, since these are expensive cryptographic operations required by several pre-quantum cryptographic algorithms and protocols. In this model, delegation algorithms are realized by a joint implementation of a computational task Alg performed with a weak, trusted client $\mathcal{T}$ together with a set of untrusted external servers $\mathcal{U}$. $\mathcal{T}$ makes queries to $\mathcal{U}$ in such a way that their interaction $\mathcal{T}^{\mathcal{U}}$ realizes Alg in a joint manner. The goal is to reduce the computational cost of $\mathcal{T}$ while guaranteeing the security of its inputs and outputs, and the possibility of verifying the correctness of the outputs of $\mathcal{U}$. The *one honest-but-curious program model* (HBC) introduced in [29] assumes that $\mathcal{U}$ always returns correct results, but may try to extract sensitive data. On the other hand, The *one-malicious version of a two untrusted program model* (OMTUP) assumes that $\mathcal{U}$ consists of two servers such that at most one of these two servers acts maliciously while $\mathcal{T}$ does not know which one of these two servers is malicious [29].

Isogenies and Cryptography. Many currently deployed public-key cryptographic primitives, such as encryption schemes, digital signatures, key encapsulation mechanisms (KEMs) or key exchange algorithms, are based on the infeasibility of either the factorization or discrete logarithm problems. Possible efficient implementations of Shor's algorithm [38] on large scale quantum computers could render these schemes insecure against such quantum adversaries. This threat resulted in the United States' National Institute of Standards and Technology (NIST) launching a post-quantum cryptography standardization process at the end of 2017. Of the 69 initially proposed key-establishment and signature protocols, a list of 15 main and alternate candidates (9 encryption and KEMs, 6 digital signature schemes) have progressed to the third round of scrutiny, announced in July 2020 [36].

One of these alternate candidates is the key encapsulation scheme SIKE [39] which is based on the intractability of the Supersingular Isogeny Diffie-Hellman problem (SIDH) originally proposed by Jao and De Feo [30]. Isogeny-based assumptions are first used by Stolbunov [42] in 2010, after rediscovering the previous work of Couveignes [21] from 2006 about hard homogeneous spaces, in order to propose a quantum resistant isogeny-based Diffie-Hellman-like key agreement scheme. The scheme in [42] is based on the difficulty of computing an isogeny of a smooth degree between two ordinary elliptic curves. However, Childs, Jao and Soukharev showed in 2014 that a quantum adversary can extract private keys of this scheme in subexponential time by using the fact that the endomor-

phism rings of the ordinary elliptic curves over finite fields are commutative [15]. Instead of using ordinary elliptic curves, a quantum resistant key agreement scheme is proposed in [30], using isogenies between supersingular elliptic curves over finite fields under the SIDH-assumption. The endomorphism rings of supersingular elliptic curves are not commutative, thus the attack introduced in [15] does not directly affect this scheme. Besides the key agreement scheme in [30] and SIKE [39], several other cryptographic schemes based on the supersingular elliptic curves have been recently proposed in the literature ranging from group key agreement schemes [3,26], zero-knowledge proofs of identity [30], identification and signature schemes [27] and hash functions [12,25] to verifiable delay functions [24].

Motivation. The significant advantage of isogeny-based cryptographic schemes are the much smaller key sizes when compared to their lattice- or code-based post-quantum counterparts. However, as highlighted by [1, p.14] (and also noted in [9]) "The main drawback to SIKE is that its performance (measured in clock cycles) is roughly an order of magnitude worse than many of its competitors. Much work has been done to optimize implementations, including the compressed-key version, and it is hoped that such optimizations continue". Furthermore, as pointed out in [35] (and also noted in [37]), post-quantum cryptographic schemes are especially required to also work efficiently on resource-constrained devices with highly limited processing storage, power and battery life to be able to utilize them in lightweight environments, which is highly desired for various applications requiring certain interoperability properties.

Following our initial work [37], we address this problem in this paper and study the secure and verifiable delegation of isogeny computations between supersingular elliptic curves over $\mathbb{F}_{p^2}$ in order to reduce the computational cost of resource-constrained clients requiring to utilize different isogeny-based cryptographic schemes.

Previous Work. In [37], two isogeny delegation algorithms were proposed in the HBC and OMTUP assumptions using the security model of Hohenberger and Lysyanskaya [29]. The first, ScIso, allowed to delegate the computation of any isogeny with revealed kernel, while allowing to push through hidden elliptic curve points or multiply unprotected points with hidden scalars. The shrouding of points was done using lookup-tables of the form $\{(i, \ell^i P)\}_{i\in\{1,\ldots,e-1\}}$, $\{(i, \ell^i Q)\}_{i\in\{1,\ldots,e-1\}}$ for generators $\langle P, Q\rangle \in E[\ell^e]$, which allowed the efficient generation of random elliptic curve points from the same torsion group. The second algorithm, HIso, used ScIso as a subroutine and allowed to hide the kernel and the codomain of the delegated isogeny. ScIso was used to reduce the cost of Jao and DeFeo's identification protocol [30] to $6\ldots14\%$ of the cost of local computation for the NIST level-2 prime $p503$ using the OMTUP assumption (where the range depends on if the random points are constructed from a secure subset or from the entire torsion group). HIso allowed to delegate the second step of SIDH-like protocols (e.g. the key exchange and public-key encryption protocols from [30] or the decapsulation step in SIKE [2,39]), reducing the cost of this step

to $\sim 60\%$ of the local computation cost for the same prime using OMTUP. The work of [37] did not propose a protocol to delegate public-key computations.

Our Contributions. The main contribution of this paper is to propose two new delegation algorithms for isogeny computations using the security model of Hohenberger and Lysyanskaya [29] in the HBC and OMTUP assumptions, and to show how to apply these to different isogeny-based cryptographic protocols and computing the respective gains for the delegator. In particular,

1. We introduce a new difficulty assumption called the *decisional point preimage problem* (DPP) that is implicitly used in the identification protocol of [30]. Furthermore, we show that it reduces to the decisional supersingular product problem (DSSP) introduced in [30].
2. We extend and improve our previous work [37], which proposed the ScIso and HIso subroutines under the HBC and OMTUP assumptions:
   (a) We present a new approach to isogeny delegation called Iso, which allows to delegate isogeny computations with unprotected kernel and to push through public and hidden points. Iso does not require (random) elliptic curve point generation using lookup-tables, which eliminates the large local memory requirement of ScIso on the delegator's side, while also speeding up the delegation algorithms.
   (b) We show how to break the HIso subroutine of [37] using pairings, and discuss some new approaches to hide the codomain curve in the delegation algorithms.
   (c) Using Iso as a subroutine, we present a new delegation algorithm, IsoDetour, which allows to delegate the computation of an isogeny without revealing the kernel. This allows IsoDetour to be used to compute public keys, a question left open in [37]. The security of IsoDetour is based on the newly proposed DPP problem.

   Our algorithms based on the OMTUP assumption rely on twisted Edwards curves, while our delegation algorithms under the HBC assumption utilize the arithmetic of Montgomery curves on the Kummer line. Mapping between these curve types can be performed very efficiently, allowing seamless integration of the delegation algorithms into cryptographic protocols using either type.
3. We further introduce the concept of *delegation-friendly primes*, which allows the delegation of public key computation in the standard SIDH-setting using our proposed IsoDetour algorithm.
4. We show how to apply our algorithms to the protocols introduced in [3,12,24,25,26,27,30] and benchmark our delegation algorithms to cryptographic protocols based on the isogenies of supersingular elliptic curves over $\mathbb{F}_{p^2}$ for various standardized SIKE primes $(p434, p503, p610, p751)$ corresponding to NIST's security levels 1, 2, 3 and 5. We also indicate the necessary communication costs between the delegator and the servers. Compared to the approach of [37], Iso allows to reduce the delegator's cost in the identification protocol of [30] to about 6% of the local computation cost in

the OMTUP and 11% in the HBC assumption for $p503$, without the need of lookup-tables. On the other hand, IsoDetour allows to reduce the cost of SIDH-type public-key generation to about 20% and 30% for OMTUP and HBC, respectively.

Outline. This work is structured as follows. Section 2 revisits the technical background needed throughout this work and introduces a new hardness assumption, called the *decisional point preimage problem*. We then present our proposed isogeny delegation algorithms in the following two sections: Section 3 introduces Iso, which allows to delegate an isogeny with disclosed kernel, while pushing through hidden points. Two implementations of Iso are presented, one in the HBC and one in the OMTUP assumption. Section 4 will then introduce IsoDetour, which uses Iso as a subroutine and further allows hiding the kernel generator of the delegated isogeny. In Section 5, we apply our delegation algorithms to different isogeny-based protocols and assess the theoretical cost reduction, and compare them with benchmarks. Section 6 concludes our findings.

Acknowledgements. The authors would like to thank Frederik Vercauteren for discussions and valuable feedback during this work. We would also like Jason LeGrow for valuable discussions concerning the isogeny cost function established in equation (3). 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.

2 Background

2.1 Elliptic curves and isogenies

We work with supersingular elliptic curves over the base field $\mathbb{F}_{p^2}$ with the Frobenius trace $t_\pi = \mp 2p$, where p is a prime. The group structure of points on elliptic curves of this type is given as follows [40,43,45]:

$$E(\mathbb{F}_{p^2}) \simeq (\mathbb{Z}/(p \pm 1)\mathbb{Z})^2 \,. \tag{1}$$

In most isogeny-based cryptographic schemes, e.g. SIDH [30] and SIKE [39], the case $t_\pi = -2p$ is used. The elliptic curves with $t_\pi = 2p$ correspond to the quadratic twists of these curves, i.e. curves having the same j-invariant which become first isomorphic over $\mathbb{F}_{p^4}$. We follow this trend and simply refer to the cases $t_\pi = 2p$ simply as the twists.

We slightly abuse notation and write e.g. $P \in E$ for $P \in E(\mathbb{F}_{p^2})$. We write $E[\tau]$ to indicate the τ-torsion group on $E(\mathbb{F}_{p^2})$ for $\tau \in \mathbb{Z}$ non-zero. Torsion groups of specific points and the generators of these groups are written with the specific point as index, e.g. we write $A \in E[\tau_A]$ and $\langle P_A, Q_A \rangle = E[\tau_A]$, where we assume A to have full order, i.e. $|\langle A \rangle| = \tau_A$. We further use the shorthands $\mathbb{Z}_\tau = \mathbb{Z}/\tau\mathbb{Z}$ and e.g. $\mathbb{Z}_A = \mathbb{Z}/\tau_A\mathbb{Z}$.

Isogenies. Isogenies are homomorphisms between two elliptic curves, that are also algebraic maps [22,40]. Separable isogenies are uniquely defined by their kernel. In the cryptographic schemes treated in this work, these kernels are subgroups of torsion groups, generated by a primitive point. For example, the group generated by $A \in E[\tau_A]$,

$$\langle A \rangle = \{\lambda A | \lambda \in \mathbb{Z}_A\} \subset E[\tau_A],$$

defines the isogeny $\alpha : E \to E/\langle A \rangle$ with $\ker \alpha = \langle A \rangle$. Any other primitive point within $\langle A \rangle$ generates the same isogeny, so we can define the equivalence class

$$[A] := \{\lambda A | \lambda \in \mathbb{Z}_A, \lambda \text{ coprime to } \tau_A\}$$

of elements defining the same isogeny kernel. One can efficiently verify if two points in $E[\tau_A]$ belong to the same equivalence class by checking if they define the same isogeny or by using pairings.

Codomain curves of isogenies are generally written in index notation, e.g. $E_A = E/\langle A \rangle$, $E_{AB} = E/\langle A, B \rangle$, where the index represents (the equivalence class of) the kernel generator of the isogeny. We will represent points on elliptic curves with a superscript corresponding to the index of the elliptic curve they are defined on, e.g. if $P \in E$, then $P^A \in E_A$ and $P^{AB} \in E_{AB}$, where we assume the used map to be clear from context. The same holds for point sets, e.g. $\{P, Q\}^A = \{P^A, Q^A\} \subset E_A$.

In order to allow efficient isogeny computations between elliptic curves, torsion groups $E[\tau]$ need τ to be smooth [30]. For most cryptographic applications, we require several smooth torsion groups of approximately the same size. This can be guaranteed by choosing $p + 1 = \prod_{i=1}^{n} \tau_i$, where $\tau_i \approx \tau_j$ for all i, j and all smooth. By this choice, supersingular elliptic curves consist of the smooth torsion groups $E[\tau_i]$ for $i = 1, \ldots, n$. Each of these torsion groups is generated by two elements, $\langle P_i, Q_i \rangle = E[\tau_i]$, so any point can be written as a linear combination of these two generators.

2.2 Elliptic curve arithmetic

Computational costs. Typical elliptic curve arithmetic operations include point addition and doubling, scalar multiplications and isogeny computation. Each of these operations can be reduced to manipulations on $\mathbb{F}_{p^2}$. In this section, we establish cost estimates for these operations in terms of the cost of multiplications m of elements over $\mathbb{F}_{p^2}$. To this end, we assume that squaring on $\mathbb{F}_{p^2}$ costs $0.8m$, while additions and comparison operations are negligible in comparison. Expensive inversions are circumvented by using projective coordinates.

We denote by A and D the theoretical cost estimates of point addition and point doubling on E, respectively, by $S(\tau)$ the cost estimate of a (large) scalar multiplication of a point by a scalar in $\mathbb{Z}_\tau$ and by $I(\tau, \mu)$ the cost estimate of computing a (large) τ-isogeny, and pushing μ points through this isogeny. Since $\tau = \prod_{i=1}^{n} \ell_i^{e_i}$ is smooth, we can approximate the cost of a τ-isogeny as the sum of the costs of individual $\ell_i^{e_i}$-isogenies. These in turn are computed

using the computation strategy described in [30]. For a given kernel generator $R \in E[\ell^e]$, the goal is to compute $\phi : E \to E_R$ using the set of intermediate kernel generators $\ell^{e-i-1}R_i \in E_i[\ell]$ for $i = 0, \ldots, e-1$, where $R_{i+1} = \phi_i(R_i)$ and $\phi_i : E_i \to E_i/\langle \ell^{e-i-1}R_i \rangle$. In the end, $\phi = \phi_{e-1} \circ \cdots \circ \phi_0 : E \to E_R$. A simple and close to optimal strategy is to perform the same amount of scalar multiplications by ℓ and ℓ-isogeny maps of R_i, while trying to minimize both. In [30], this is referred to as the balanced scenario, and either operation has to be performed $\frac{e}{2}\log_2 e$ times. We refer to the cost of the former as S_ℓ and the latter as P_ℓ, both depending on ℓ. Furthermore, we have to construct exactly e codomain curves $E_1, \ldots, E_e$ for the cost C_ℓ. If we also push through additional points, we need to do this once for each curve, thus e times, also at the cost of P_ℓ. We find the cost of an ℓ^e-isogeny to be

$$I(\ell^e, \mu) = (P_\ell + S_\ell)\frac{e}{2}\log_2 e + (C_\ell + \mu P_l)e\,. \tag{2}$$

Note that we omit the cost of kernel generation as we will consider that separately. Computing a τ-isogeny, where $\tau = \prod_{i=1}^{n} \ell_i^{e_i}$ amounts to n consecutive ℓ_i-isogenies for $i = 1, \ldots, n$. We also push through the generators of each of these torsion groups, which amounts to evaluating each $\ell_i^{e_i}$-isogeny $n - i$ more times for $i = 1, \ldots, n-1$. Finally, we find the cost of a τ-isogeny

$$I(\tau, \mu) = \sum_{i=1}^{n}\left[(P_{\ell_i} + S_{\ell_i})\frac{e_i}{2}\log_2 e_i + (C_i + \mu P_i)e_i\right] + \sum_{i=1}^{n-1} P_{\ell_i}e_i(n-i)\,. \tag{3}$$

Note that recent results in [5] reduce the asymptotic complexity of ℓ-isogeny computations from $\tilde{O}(\ell)$ to $\tilde{O}(\sqrt{\ell})$. However, this advantage only manifests itself from $\ell \simeq 100$ upwards. Even though we will be working with torsion groups τ that potentially have $\max_i\{\ell_i\} > 100$, these τ-isogeny computations will only be performed on the server-side. The cryptographic protocols in this work will generally be based on very small primes $\ell \in \{2, 3, 5, \ldots\}$, unless stated otherwise.

Montgomery curves. Montgomery curves are elliptic curves of the form

$$E_{a,b} : bY^2Z = X^3 + aX^2Z + XZ^2\,,$$

with $b \neq 0$ and $a^2 \neq 4$. Arithmetic operations and isogeny computations on Montgomery curves are particularly efficient if they are reduced to the Kummer line $E/\langle \pm 1 \rangle$ by mapping out the Y-coordinate and reducing points to the X and Z coordinates [20,34]. Points on the Kummer line form no longer a group, and addition operations have to be substituted by differential additions, which require 3 inputs (P, Q and $P - Q$) to compute $P + Q$. Note that arithmetic operations and isogeny computations are independent of the parameter b [18]. In fact, changing b only allows to move between a curve and its isomorphisms or its quadratic twist, the latter being unified on the Kummer line. Thus, working on Montgomery curves does not only have the advantage of efficient arithmetic, but it also allows to easily switch between isomorphic curves and quadratic twists without the requirement of working in extension fields [16].

Using the results from [7,20], the cost of point addition and doubling on $E(\mathbb{F}_{p^2})$ can be estimated by

$$A = 5.6m \quad \text{and} \quad D = 3.6m\,,$$

respectively. Scalar multiplications are performed in $\lceil \log_2 \tau \rceil$ steps of the Montgomery ladder algorithm [34] for the total estimated cost of [37]

$$S(\tau) = M\lceil \log_2 \tau \rceil - A\,, \quad \text{where} \quad M = A + D = 9.2m\,. \tag{4}$$

Using the optimized results from [17], we find

$$\begin{aligned} C_3 &= 4.4m, \quad S_3 = 9.2m, \quad P_3 = 5.6m, \\ C_4 &= 3.2m, \quad S_4 = 7.2m, \quad P_4 = 7.6m, \end{aligned}$$

for the parameters in the isogeny computation (3).

Twisted Edwards curves. Twisted Edwards curves are elliptic curves of the form

$$E_{c,d} : cX^2Z^2 + Y^2Z^2 = Z^4 + dX^2Y^2\,,$$

with $d \neq 0, 1$ and $c \neq 0$. While there are many coordinate representations in [6], the extended coordinates introduced in [28] are of particular interest for this work. In [28], point coordinates are extended by a fourth element $T = XY/Z$, which allows for efficient arithmetic with

$$A = 9m \quad \text{and} \quad D = 7.2m\,.$$

If $-c$ is a square in $\mathbb{F}_{p^2}$, unsing the isomorphism $X \to X/\sqrt{-c}$ results in an even more efficient addition of $A = 8m$. Scalar multiplications can be performed using the double-and-add method, bound by the cost

$$S(\tau) = M\lceil \log_2 \tau \rceil - A\,, \quad \text{where} \quad M = A + D = 16.2m\,. \tag{5}$$

While the arithmetic is noticeably slower than the one on Montgomery curves, points on twisted Edwards curves form a group and allow more versatile constructions. For an overview of isogeny-related costs on twisted Edwards curves we refer to [4].

Mapping between Montgomery and Twisted Edwards curves. There is a one-to-one correspondence between Montgomery and twisted Edwards curves [6], and switching between equivalent curves can be done using the following maps

$$c = \frac{a+2}{b}, \quad d = \frac{a-2}{b} \quad \text{and} \quad a = \frac{c+d}{c-d}, \quad b = \frac{4}{c-d}\,.$$

By writing the curve parameters in projective coordinates, we avoid inversions and reduce these maps to simple additions on $\mathbb{F}_{p^2}$. To map points between these curves, we can use the maps [6,10,33]

$$\begin{aligned} (X_M : Z_M) &= (Z_E + Y_E : Z_E - Y_E) \\ (X_E : Y_E : Z_E) &= (X_M(X_M + Z_M) : Y_M(X_M - Z_M) : Y_M(X_M + Z_M))\,, \end{aligned}$$

while $T_E = X_E Y_E / Z_E = X_M(X_M - Z_M)$ can be easily computed from the other coordinates. While the point map from Montgomery to twisted Edwards curves is given by two finite field additions (assumed to have negligible cost), the inverse way can be computed in $3m$, or in $3.8m$ if T_E is computed as well.

Communication cost. In order to give an idea of the amount of data exchanged between the delegator and the server, we express their communication costs in bits. Let $b(p) = \lceil \log_2 p \rceil$ denote the amount of information in a $\log_2 p$-bit number. Elements in $\mathbb{F}_{p^2}$ then contain $2b(p)$ bits of information. We note that elliptic curves in Montgomery form are fully defined by the parameter $a \in \mathbb{F}_{p^2}$ and that points can be represented on their Kummer line, i.e. as $P = (X : Z)$, where $X, Z \in \mathbb{F}_{p^2}$. If $Z = 1$, which can always be achieved by a single inversion, a point can completely be expressed by its X-coordinate. Thus, in most cases (unless stated otherwise), both points and elliptic curves contain $2b(p)$ bits of information. For twisted Edwards curves, we need both curve parameters and points are expressed using four elements in $\mathbb{F}_{p^2}$. By setting $Z = 1$, we can reduce this to two elements, X and Y, and recover T by a simple multiplication. Then, both points and elliptic curves each contain $4b(p)$ bits of information. In the case $p \approx \prod_{i=1}^{n} \tau_i$ with $\forall i, j : \tau_i \approx \tau_j$, elements in $\mathbb{Z}_{\tau_i}$ can be expressed using approximately $b(p)/n$ bits.

2.3 Security model

The security model for delegating cryptographic computations used throughout this paper was originally proposed by Hohenberger and Lysyanskaya [29]. In this model, delegation algorithms are split into a trusted component $\mathcal{T}$ and a set of untrusted servers $\mathcal{U}$. The delegator makes oracle queries to the servers such that their interaction $\mathcal{T}^{\mathcal{U}}$ results in the correct execution of an algorithm Alg with the goal of reducing the computational cost of $\mathcal{T}$ when compared to the local execution of Alg. Since $\mathcal{U}$ might potentially be malicious, the delegator needs to both ensure that $\mathcal{U}$ is not able to extract any sensitive data from the interaction, and be able to verify that the results returned by $\mathcal{U}$ are computed correctly. The full adversary in this model $\mathcal{A} = (\mathcal{E}, \mathcal{U})$ further includes the environment $\mathcal{E}$, representing any third party, that should also not be able to extract sensitive data, while having a different view of the inputs and output of Alg as $\mathcal{U}$ does.

The *outsource input/output specification* (or *outsource-IO*) distinguishes *secret* (only $\mathcal{T}$ has access), *protected* ($\mathcal{T}$ and $\mathcal{E}$ have access) and *unprotected* (everyone has access) inputs and outputs, while non-secret inputs are further subdivided into *honest* and *adversarial*, depending on whether they originate from a trusted source or not.

An important assumption of this model is that, while the servers in $\mathcal{U}$ and the environment $\mathcal{E}$ might initially devise a joint strategy, there is no direct communication channel between the different servers within $\mathcal{U}$ or between $\mathcal{U}$ and the environment $\mathcal{E}$ after $\mathcal{T}$ starts using them. However, they could try to establish an indirect communication channel via the unprotected inputs and un/protected outputs of Alg. To mitigate this threat, $\mathcal{T}$ should ensure that the adversarial,

unprotected input stays empty (see also Remark 2.4 in [29]), while the non-secret outputs do not contain any sensitive data. This is formalized in the following definition:

Definition 1 (Outsource-security). *[29] Let* Alg *be an algorithm with outsource-IO. The pair* $(T,\mathcal{U})$ *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.*
  - ***Pair One****:* $\mathcal{E}VIEW_{real} \sim \mathcal{E}VIEW_{ideal}$ *(*$\mathcal{E}$ *learns nothing.)*
  - ***Pair Two****:* $\mathcal{U}VIEW_{real} \sim \mathcal{U}VIEW_{ideal}$ *(*$\mathcal{U}$ *learns nothing.)*

  The details of these experiments can be found in Definition 2.2 of [29]. If $\mathcal{U}$ *consists of multiple servers, then there is a PPT-simulator* $\mathcal{S}_{2,i}$ *for each of their views.*

Adversarial models differ along the number and intent of servers. The models we will analyze in this work are the following.

Definition 2 (Honest-but-curious [14]). *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 3 (OMTUP [13,29,44]). *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 refer to the paper of Hohenberger and Lysyanskaya [29] for other security models without any honest party, namely the *two untrusted program model* (TUP) and the *one untrusted program model* (OUP). We discuss models without honest entity in Section 3.3.

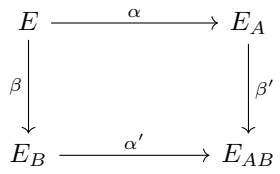
We formalize $\mathcal{T}$'s efficiency gain due to the delegation, as well as its ability to verify correctness of $\mathcal{U}$'s outputs in the following definition.

Definition 4 ((α,β)-outsource-security [29]). *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* $\mathsf{Alg}(x)$*,*
- *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* $\geq \beta$*.*

2.4 Cryptographic protocols and difficulty assumptions

Let E be a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$. Cryptographic protocols in the SIDH setting are generally based on the following commutative diagram:



In general, E is a publicly known starting curve with at least two coprime torsion groups $\langle P_A, Q_A \rangle = E[\tau_A]$ and $\langle P_B, Q_B \rangle = E[\tau_B]$, whose generators are also publicly known. Let $\langle A \rangle = \ker \alpha$ and $\langle B \rangle = \ker \beta$ and let E_A and E_B be the respective codomains of α and β. Then the commutativity of the upper diagram is given by choosing $\ker \alpha' = \langle A^B \rangle$ and $\ker \beta' = \langle B^A \rangle$.

These commutative diagrams can then used to build cryptosystems. For instance, the key agreement protocol SIDH [30] assigns $E[\tau_A]$ to Alice and $E[\tau_B]$ to Bob. After choosing $A = P_A + aQ_A \in E[\tau_A]$ for a secret $a \in \mathbb{Z}_A$, Alice computes and publishes her public key (E_A, P_B^A, Q_B^A), while Bob does the same (E_B, P_A^B, Q_A^B) for $B = P_B + bQ_B \in E[\tau_B]$ and $b \in \mathbb{Z}_B$. Both can then compute the shared public key E_{AB} by computing α' from $A^B = P_A^B + aQ_A^B \in E_B[\tau_A]$ or β' from $B^A = P_B^A + bQ_B^A \in E_A[\tau_B]$. In a similar way, an identification protocol is proposed in [30]: the owner of a public key (E_A, P_B^A, Q_B^A) can prove its knowledge of the secret a that generates it. This is done by the prover committing to (E_B, E_{AB}) for a random $B \in E[\tau_B]$, and then, depending on the received challenge $c \in \{0, 1\}$, either revealing (B, B^A) or A^B. This interaction is repeated until the desired soundness is reached.

We revisit some of the security assumptions upon which isogeny-based cryptographic protocols are based. Note that we only show the ones that are explicitly used in this work. For other hard problems, we refer for example to [30].

Problem 1 (Computational Supersingular Isogeny Problem (CSSI) [30]). Given the triplet (E_B, P_A^B, Q_A^B), find an element in $[B] \subset E[\tau_B]$.

Problem 2 (Decisional Supersingular Product Problem (DSSP) [30]). Let $\alpha : E \to E_A$. Given a tuple $(E, E_A, E_1, E_2, \alpha, \alpha')$, determine from which of the following distributions it is sampled

- E_1 is a random curve with $|E| = |E_1|$ and $\alpha' : E_1 \to E_2$ is a random τ_A-isogeny,
- $E_1 \times E_2$ is chosen at random among those isogenous to $E \times E_A$ and where $\alpha' : E_1 \to E_2$ is a τ_A-isogeny.

We further define the following difficulty assumption and show that it is at least as hard as DSSP.

Problem 3. **Decisional Point Preimage Problem (DPP)** Let $\beta : E \to E_B$. Given (E, E_B, A, A'^B), where $A \in E[\tau_A]$, and $A'^B \in E_B[\tau_A]$, decide whether $[A] = [A']$.

Theorem 1. *Let $\mathcal{A}_{DPP}$ be an adversary against the Decisional Point Preimage Problem. Then, $\mathcal{A}_{DPP}$ can be used as a subroutine against the Decisional Supersingular Product Problem.*

Proof. Let $\mathcal{A}_{\mathrm{DPP}}$ be an adversary to the DPP problem which, upon receiving the tuple (E, E_B, A, A'^B), returns $b = 1$ if $[A^B] = [A'^B]$ and $b = 0$ otherwise. Then, we can construct an adversary $\mathcal{B}_{\mathrm{DSSP}}^{\mathcal{A}_{\mathrm{DPP}}}$ against DSSP, which returns $b = 0$ in the first and $b = 1$ in the second case of Problem 2. Upon receiving $(E, E_A, E_B, E_C, \alpha, \alpha')$, $\mathcal{B}_{\mathrm{DSSP}}^{\mathcal{A}_{\mathrm{DPP}}}$ extracts kernel generators $\langle S \rangle = \ker \alpha$ and $\langle S'^B \rangle = \ker \alpha'$, then sends the query (E, E_B, S, S'^B) to $\mathcal{A}_{\mathrm{DPP}}$. $\mathcal{B}_{\mathrm{DSSP}}^{\mathcal{A}_{\mathrm{DPP}}}$ returns what $\mathcal{A}_{\mathrm{DPP}}$ returns, since if $[S] = [S']$, then $E_B \times E_C$ is isogenous to $E \times E_A$ and we have $b = 1$, otherwise $b = 0$. □ □

3 Delegating isogenies

Throughout this section, we assume that the delegator $\mathcal{T}$ is able to generate elements in $\mathbb{Z}$ uniformly at random in an efficient manner. We further assume that $\mathcal{T}$ knows a representation of any of its secret and protected points in terms of the public torsion group generators.

3.1 Advertised server functionality

Let E be an elliptic curve and $\mathcal{K}, \mathcal{M} \subset \mathbb{Z} \times E(\mathbb{F}_{p^2})$ two distinct sets of scalar-point pairs. The first set, $\mathcal{K}$, represents the kernel of the delegated isogeny, i.e. such that $K = \sum_{(a,P)\in\mathcal{K}} aP$ is the kernel generator. The second set, $\mathcal{M}$, contains points to be pushed through this isogeny and multiplied with the associated scalar. Thus, given an input $(E, \mathcal{K}; \mathcal{M})$ from the delegator, a server first determines the kernel generator K, then uses it to define the isogeny $\phi : E \to E_K$ and finally the computes the set $\mathcal{M}^K := \{aP^K \mid (a, P) \in \mathcal{M}\}$, where $P^K = \phi(P)$. The server then returns $(E_K; \mathcal{M}^K)$. We write the delegation step as follows

$$(E_K; \mathcal{M}^K) \leftarrow \mathcal{U}(E, \mathcal{K}; \mathcal{M}).$$

Note that the case $\mathcal{K} = \emptyset$ results in $\mathcal{U}$ performing scalar multiplications. In the case of multiple servers $(\mathcal{U}_i)_{i\in\{1,\ldots,n\}}$, we will distinguish the different $\mathcal{M}_i$ and $\mathcal{K}_i$ using the appropriate index. In order to reduce the communication cost, we assume that servers return all points scaled with $Z = 1$.

Notation. For pairs in either $\mathcal{K}_i$ or $\mathcal{M}_i$, if the scalar is 1, we omit it and write e.g. P instead of $(1, P)$. If the kernel set $\mathcal{K}$ consists of a single such point, we further drop the set brackets. Multiple pairs with the same point are merged, e.g. we write $\{(a, P), (b, P), (c, P)\}$ as $\{(\{a, b, c\}, P)\}$.

3.2 The Iso-Algorithm

We propose our first isogeny delegation algorithm Iso:

Definition 5 (The Iso-algorithm). *The* isogeny delegation algorithm Iso *takes as inputs a supersingular elliptic curve $E/\mathbb{F}_{p^2}$, a kernel set $\mathcal{K} \subset \mathbb{Z} \times E(\mathbb{F}_{p^2})$, and two scalar-point pair sets $\mathcal{H}_0, \mathcal{H} \subset \mathbb{Z} \times E(\mathbb{F}_{p^2})$, then computes the isogeny $\phi : E \to E_K$ and produces the output $(E_K; \mathcal{H}_0^K, \mathcal{H}^K)$, where $K = \sum_{(a,P)\in\mathcal{K}} aP$ and $\mathcal{H}_{(0)}^K = \{aP^K \mid (a, P) \in \mathcal{H}_{(0)}\}$. The inputs $E, \mathcal{K}, \mathcal{H}_0$ are all honest, unprotected parameters, while $\mathcal{H}$ contains secret or (honest/adversarial) protected scalars and honest, unprotected points. The outputs E_K and $\mathcal{H}_0^K$ are unprotected while $\mathcal{H}^K$ is secret or protected. We write*

$$(E_K; \mathcal{H}_0^K, \mathcal{H}^K) \leftarrow \mathsf{Iso}(E, \mathcal{K}; \mathcal{H}_0, \mathcal{H}).$$

In Figures 1 and 2, we show how a delegator $\mathcal{T}$ can use the advertised server functionality from Section 3.1 in order to implement Iso in an outsource-secure way under the HBC and OMTUP assumptions. The delegation subroutines are organized according to 5 main steps:

- Gen: Generation of auxiliary elements
- Shr: Shrouding of hidden elements using the auxiliary elements
- Del: Delegation to the server
- Ver: Result verification
- Out: Result recovery and output

Note that the HBC case does not need a verification step by assumption. The idea behind Figure 1 is relatively trivial but effective: the delegator hides the secret/protected scalars simply by not disclosing them to the server and computing the scalar multiplication on the codomain point itself.

Honest-but-curious approach.

- Gen: No auxiliary elements are needed.
- Shr: Set $\mathcal{H}' = \{Q \mid (a, Q) \in \mathcal{H}\}$.
- Del: Delegate $(E_K; \mathcal{H}_0^K \cup \mathcal{H}'^K) \leftarrow \mathcal{U}(E, \mathcal{K}; \mathcal{H}_0 \cup \mathcal{H}')$.
- Out: Compute $\mathcal{H}^K = \{aQ^K \mid (a, Q) \in \mathcal{H}, Q^K \in \mathcal{H}'^K\}$, then return $(E_K; \mathcal{H}_0^K, \mathcal{H}^K)$.

Fig. 1. Implementation of Iso in the HBC assumption

The OMTUP case of Figure 2 is a bit more complex, but will result in a lower cost for the delegator when compared to the HBC case. The underlying idea (for $N = 1$) is that the delegator shrouds the secret/protected scalars as a linear combination of small and large random scalars. The large scalars are distributed between the two servers in order to prevent reconstruction of the secrets, while

the small scalars are kept secret by the delegator and used to ultimately verify correctness of the returned points. The size of the small scalars influences the cost for the delegator and the verifiability of the protocol. To further increase verifiability, the delegator can add more random scalars to the mix by increasing N, which leads to multiple, interconnected verification conditions, and results in an even higher verifiability, albeit at a higher cost for the delegator. There is an optimal trade-off between these two parameters, depending on the desired verifiability. We will discuss this trade-off further in Section 3.2.

OMTUP approach.

Gen: For each $(a, Q) \in \mathcal{H}$, choose $N \in \mathbb{N}$, then (assuming $Q \in E[\tau]$) generate
- small non-zero scalars $c_1, \ldots, c_N, d_1, \ldots, d_N \in \{-2^{t-1}, \ldots, 2^{t-1}\}$, and
- random scalars $r_0, s_0, s_1, \ldots, s_{N-1} \in \mathbb{Z}_\tau$.

Shr: For each $(a, Q) \in \mathcal{H}$, compute $r_i = -s_i + c_i s_0 + d_i r_0$ for $i = 1, \ldots, N-1$. Define $\sigma = \sum_{i=1}^{N-1}(s_i + r_i)$ and let γ be the smallest integer > 1 coprime to τ, then compute $s_N = \gamma^{-1}(d_N r_0 + c_N s_0 + \sigma - a)$ and $r_N = -s_N + c_N s_0 + d_N r_0$. Set

$$\mathcal{H}_1' = \{(\{s_0, \ldots, s_N\}, Q) \mid (a, Q) \in \mathcal{H}\}, \quad \mathcal{H}_2' = \{(\{r_0, \ldots, r_N\}, Q) \mid (a, Q) \in \mathcal{H}\},$$

with the scalars in random order.

Del: Delegate

$$(E_K; \mathcal{H}_0^K \cup \mathcal{H}_1'^K) \leftarrow \mathcal{U}_1(E, \mathcal{K}; \mathcal{H}_0 \cup \mathcal{H}_1')$$
$$(E_K'; \mathcal{H}_0'^K \cup \mathcal{H}_2'^K) \leftarrow \mathcal{U}_2(E, \mathcal{K}; \mathcal{H}_0 \cup \mathcal{H}_2').$$

Ver: Verify, if $E_K \stackrel{?}{=} E_K'$ and $\mathcal{H}_0^K \stackrel{?}{=} \mathcal{H}_0'^K$, and for $i = 1, \ldots, N$, if

$$(s_i Q)^K + (r_i Q)^K \stackrel{?}{=} c_i (s_0 Q)^K + d_i (r_0 Q)^K.$$

Out: If any of the verifications fail, return $\perp$, otherwise compute

$$\mathcal{H}^K = \left\{ r_N Q^K - (\gamma - 1) s_N Q^K + \sum_{i=1}^{N-1} (s_i Q^K + r_i Q^K) \,\middle|\, (a, Q) \in \mathcal{H} \right\}$$

and return $(E_K; \mathcal{H}_0^K, \mathcal{H}^K)$.

Fig. 2. Implementation of Iso in the OMTUP assumption

Hiding a point. If the delegator wants to push through a secret or (honest/adversarial) protected elliptic curve point $A = P + aQ \in E[\tau_A]$, then $\mathcal{T}$ simply has to delegate

$$(E_K; \mathcal{H}_0^K \cup \{P\}, \mathcal{H}^K \cup \{aQ^K\}) \leftarrow \mathsf{Iso}(E, \mathcal{K}; \mathcal{H}_0 \cup \{P\}, \mathcal{H} \cup \{(a, Q)\}),$$

and compute $A^K = P^K + aQ^K$. We assume that a representation of A in the normal form is always known, as will always be the case in the cryptographic

protocols that we discuss in this paper. We will generally write

$$(E_K; \mathcal{H}_0^K, \mathcal{H}^K \cup \{A^K\}) \leftarrow \mathsf{Iso}(E, \mathcal{K}; \mathcal{H}_0, \mathcal{H} \cup \{A\})$$

for the sake of conciseness. This notation also suggests itself due to the same verifiability for the recovery of A^K as for aQ^K, since P is fully verifiable. Nevertheless, we have to pay attention to the fact that the cost in the recovery step increases by one elliptic curve addition and that $\mathcal{H}_0$ implicitly contains P.

Outsource-security in the HBC case We analyze some properties of the protocol in Figure 1 leading to the following theorem.

Theorem 2. *Under the honest-but-curious assumption, the outsourcing algorithm $(\mathcal{T},\mathcal{U})$ given in Figure 1 is a $\left(O\left(\frac{1}{\log\log\tau}\right), 1\right)$-outsource secure implementation of Iso, where τ is the smooth degree of the delegated isogeny.*

Correctness. Correctness is given by the homomorphism property of isogenies.

Cost. We define $\mu_0 = \#\mathcal{H}_0$ and $\mu = \#\mathcal{H}$. For every point in $\mathcal{H}$, we have to compute one scalar multiplication at the cost of $S(\tau_A)$, for $Q \in E[\tau_A]$. Assuming the different torsion groups have approximately the same size (which will always be the case in our delegation schemes), we find $T_{\mathrm{HBC}}(\mu, \tau_A) = \mu S(\tau_A)$. The local computation, assuming $K \in E[\tau_K]$ and $\tau_K = \sum_i \ell_i^{e_i}$ on the other hand would amount to $I(\tau_K, \mu_0 + \mu)$, so that we find

$$\alpha_{\mathrm{HBC}}(\mu_0, \mu, \tau_A, \tau_K) = \frac{\mu S(\tau_A)}{I(\tau_K, \mu_0 + \mu)} = \frac{\mu(M\lceil \log_2 \tau_A \rceil - A)}{\sum_i (I_{\ell_i} + S_{\ell_i} + (\mu_0 + \mu)P_{\ell_i})\frac{e_i}{2}\log_2 e_i}$$

Assuming μ small and $\tau_A \approx \tau_K$,[3] we find

$$\alpha_{\mathrm{HBC}}(\mu_0, \mu, \tau_A, \tau_K) = O\left(\frac{\log \tau_A}{\log \tau_K \log\log \tau_K}\right) = O\left(\frac{1}{\log\log \tau_K}\right)$$

Security. Neither $\mathcal{U}$ nor $\mathcal{E}$ can gain any information about the hidden parameters a, due to the fact that they are never disclosed in any form. In every round, the simulators $\mathcal{S}_1$ and $\mathcal{S}_2$ simply proceed as in the real execution of the protocol. Therefore $\mathcal{EVIEW}_{\mathrm{real}} \sim \mathcal{EVIEW}_{\mathrm{ideal}}$ and $\mathcal{UVIEW}_{\mathrm{real}} \sim \mathcal{UVIEW}_{\mathrm{ideal}}$.

Outsource-security in the OMTUP case We analyze some properties of the protocol in Figure 2 leading to the following theorem.

Theorem 3. *Under the OMTUP assumption, the outsourcing algorithm $(\mathcal{T}, (\mathcal{U}_1, \mathcal{U}_2))$ given in Figure 2 is an $\left(O\left(\frac{t}{\log\tau\log\log\tau}\right), 1 - \frac{1}{(N+1)2^{Nt}}\right)$-outsource secure implementation of Iso, where τ is the smooth degree of the delegated isogeny. If $\mathcal{H} = \emptyset$, then it is fully verifiable.*

[3] Note that we can easily assume $S(\tau_A) \approx S(\tau_K)$ for $\tau_A \approx \tau_K$. On the other hand, this approximation does not in general hold for $I(\tau_A, n)$ and $I(\tau_K, n)$, so that we substitute $\tau_A \to \tau_K$ in our formula and not the other way around.

Correctness. Since for $i = 1, \ldots, N$, we have $r_i = -s_i + c_i s_0 + d_i r_0$, the verification conditions hold due to the homomorphism property of isogenies. At the end, the delegator returns for each $(a, Q) \in \mathcal{H}$

$$\begin{aligned} r_N Q^K - (\gamma - 1) s_N Q^K + \sum_{i=1}^{N-1} (s_i Q^K + r_i Q^K) \\ &= ((r_N + s_N) - \gamma s_n + \sigma) Q^K \\ &= ((c_N s_0 + d_N r_0) - \gamma\gamma^{-1}(c_N s_0 + d_N r_0 + \sigma - a) + \sigma) Q^K \\ &= a Q^K. \end{aligned}$$

Verifiability. The malicious server can not successfully return a wrong codomain curve or map of $\mathcal{H}_0$, since the delegator compares it to the honest server's result. Yet, the malicious server could try to cheat on the point maps in $\mathcal{H}$. The only options are points that still satisfy all the verification conditions. Assume without loss of generality that $\mathcal{U}_1$ is the malicious server and wants to return a wrong $s_1 Q^K$ by shifting it with another point X_1, i.e. it returns $s_1 Q^K + X_1$ instead. In order for the according verification condition to still hold, the server will also have to shift $S = s_0 Q^K$ by some point Y. Thus, in order for the verification condition to still hold, the malicious server has to guarantee that

$$s_1 Q^K + X_1 - c_1(s_0 Q^K + Y) = s_1 Q^K - c_1 s_0 Q^K, \quad \text{implying} \quad X_1 - c_1 Y = 0.$$

Since the server does not know c_1 it has to guess it, which it only can do with a probability of at most 2^{-t}. Furthermore, by shifting S by Y, all the other verification conditions have to be rectified as well, i.e. $X_i - c_i Y = 0$ for all $i = 1, \ldots, N$. In order to be successful, the malicious server is thus required to solve the linear system

$$\begin{pmatrix} 1 & & -c_1 \\ & \ddots & \vdots \\ & & 1 \; -c_N \end{pmatrix},$$

which can only be done by guessing all $c_1, \ldots, c_N$ correctly. As a final challenge, the server also has to identify which of the (at least) $N + 1$ scalars given to it corresponds to s_0. Hence, the probability of a malicious server succeeding with this attack for $Q \in E[\tau_A]$ is thus bounded by $(N + 1)^{-1} 2^{-Nt}$.

Cost. We again define $\mu_0 = \#\mathcal{H}_0$ and $\mu = \#\mathcal{H}$. Since the points in $\mathcal{H}$ are shrouded independently, we derive the delegator's cost per point in $\mathcal{H}$. In the shrouding step, the delegator has to compute $2m$ per r_i and another $3m$ for s_N, thus $2Nm + 3m$, assuming γ^{-1} is known on the underlying field. In the verification step, the delegator has to compute $2NA$ as well as $2N$ scalar multiplications. In order to compare the points on both sides of the verification equation, we need to scale them to the same Z-value. This can be achieved by multiplying each side with the opposing Z coordinate for a total cost of $2Nm$.

Finally in the output step the delegator, already knowing the terms $s_iQ^K + r_iQ^K$ for $i = 1, \ldots, N-1$ from the verification step, is left to compute $NA + S(\gamma - 1)$. In the cryptosystems used throughout this work, $\gamma \in \{2, 3\}$, so that the cost of the verification step is bound by $(N+1)A$.

Naively, the scalar multiplications in the verification step would cost $2NS(2^t)$, but we can easily decrease this by realizing that all of the scalar multiplications, which the delegator has to perform, are multiples of the same two points, i.e. s_0Q^K and r_0Q^K. Thus we need to perform the doubling part of the double-and-add algorithm only once per point. We can even go further and reduce the total addition part for the different N by grouping the repeating patterns.

To this end, assume we want to compute the multiplications $c_1Q, \ldots, c_NQ$ for the scalars $c_i = (c_{i,0} \ldots c_{i,t-1})_2$ and for any point Q. We first ignore the sign of the c_i and define the sets $C_i = \{j \mid c_{i,j} = 1\}$ and the index power set $S = \mathcal{P}(\{1, \ldots, N\})$, excluding the empty set. For each $k \in S$, let

$$A_k = \left(\bigcap_{i \in k} C_i\right) \Bigg\backslash \left(\bigcup_{j \notin k} C_j\right)$$

enumerate all possible distinct areas in the Venn diagram of those sets. Then, after having computed all the doubling operations $2Q, \ldots, 2^{t-1}Q$, the delegator computes the sums

$$Q_k = \sum_{j \in A_k} 2^j Q \tag{6}$$

for each $k \in S$, then adds the appropriate sets for each c_i, i.e.

$$c_iQ = \sum_{k \in S_i} Q_k \,, \tag{7}$$

where $S_i = \{k \in S \mid i \in k\}$. Finally, the delegator applies the correct sign to the result of (7).

With this in mind, we can express the maximal number of point additions in these two steps. Let $\omega = \#S = 2^N - 1$ and $\omega_i = \#S_i = 2^{N-1}$ and let ω_0 denote the number of empty sets A_k. We assume $t > \omega$, which will later be guaranteed by our choices of N and t. Since $|\bigcup_k A_k| \leq t$ and $A_k \cap A_{k'} = \emptyset \ \forall k \neq k'$, the number of additions the delegator has to perform in order to compute the different Q_k in (6) is at most t, reduced by the number of non-empty sets (the "missing" additions will later be done between those sets) thus $t - (\omega - \omega_0)$.

In (7), the delegator can use a tree structure to add the different sets. It is easy to verify, that $|\bigcap_{i=1}^m S_i| = 2^{-m}|S_i| = 2^{N-1-m}$. If the delegator chooses the order of the sum in (7), for e.g. c_1Q, in such a way as to start with all the elements in $S_1 \cap S_2$, then it has already computed half of c_2Q too, so that in the first step it needs $2^{N-1} - 1$ additions while in the second, only $1 + (2^{N-2} - 1)$ are left. If we start the computation of c_1Q with $S_1 \cap S_2 \cap S_3$, then $S_1 \cap S_2$, then $S_2 \cap S_3$ and the computation of c_2Q with $S_2 \cap S_3$, we can compute c_3Q with only $3 + 2^{N-3} - 1$ additions. Proceeding similarly with further computations, we

find that for the computation of c_iQ, $\mathcal{T}$ needs to perform

$$(2^{i-1}-1)+(2^{N-i}-1)$$

additions. Summing over $i=1,\ldots,N$, we find that $\mathcal{T}$ has to add at most $2(2^N-N-1)-\omega_0$ sets, after subtracting the number of empty sets. Including the initial doubling operations, we find the total maximal cost of

$$S_N(t)=Mt+(2^N-2N-2)A$$

for N parallel scalar multiplications c_iQ, where $M=D+A$.

Remark 1. At this point, we would like to note that this approach is only possible for points in a group and is in particular not realizable on the Kummer line of Montgomery curves. Furthermore, on Montgomery curves, the delegator could not even compute e.g. s_iQ+r_iQ, using differential addition, since he would be lacking the knowledge of $(s_i-r_i)Q$, which we can't get from one of the servers without revealing information.

We can finally express[4]

$$\begin{aligned}T_{\mathrm{OMTUP}}(\mu,t)&=\mu\left[(4N+3)m+3NA+A+2S_N(2^t)\right]\\&=\mu\left[(4N+3)m+2Mt+(2^{N+1}-N-3)A\right]\ .\end{aligned}$$

The cost reduction can then be expressed as follows

$$\alpha_{\mathrm{OMTUP}}(\mu_0,\mu,t,\tau)=\frac{T_{\mathrm{OMTUP}}(\mu,t)}{I(\tau,\mu_0+\mu)}=\frac{\mu\left[(4N+3)m+2Mt+(2^{N+1}-N-3)A\right]}{\sum_i(I_{\ell_i}+S_{\ell_i}+(\mu_0+\mu)P_{\ell_i})\frac{e_i}{2}\log_2 e_i}$$

Assuming N is fixed and since $t>2^N-1$, the dominating term in the numerator is $2\mu Mt$. Dropping scalar factors and assuming μ,μ_0 to be small, we find

$$\alpha_{\mathrm{OMTUP}}(t,\tau)=O\left(\frac{t}{\log\tau\log\log\tau}\right)\ .$$

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. We reduce our analysis to a single pair $(a,Q)\in\mathcal{H}$ as it extends naturally to multiple hidden scalars. We assume $a\in\mathbb{Z}_\tau$ and $Q\in E[\tau]$.

– **Pair One:** $\mathcal{EVIEW}_{\mathrm{real}}\sim\mathcal{EVIEW}_{\mathrm{ideal}}$
If the input a is not secret, $\mathcal{S}_1$ simply behaves as in the real execution of the protocol. If a is secret, then in round i, $\mathcal{S}_1$ generates $2(N+1)$ random scalars $u_0,\ldots,u_N,v_0,\ldots,v_N$ and makes the queries:

$$\begin{aligned}(E_K,\mathcal{H}_0^K\cup\mathcal{H}_1'^K)&\leftarrow\mathcal{U}_1(E,\mathcal{K},\mathcal{H}_0\cup\{(\{u_0,\ldots,u_N\},Q)\}),\\(E_K',\mathcal{H}_0'^K\cup\mathcal{H}_2'^K)&\leftarrow\mathcal{U}_2(E,\mathcal{K},\mathcal{H}_0\cup\{(\{v_0,\ldots,v_N\},Q)\}).\end{aligned}$$

[4] We omit N from the function's input parameters as it depends on the choice of t, as will be shown in Section 3.2.

to the servers, then verifies if all the outputs are correct. If either $E_K, E'_K, \mathcal{H}_0, \mathcal{H}'_0$ is incorrect, $\mathcal{S}_1$ returns $(Y_p^i, Y_u^i, replace^i) = (\perp, \emptyset, 1)$. If either of the results in $\mathcal{H}_1^K$ or $\mathcal{H}_2^K$ is incorrect, then with probability $(N+1)^{-1}2^{-Nt}$, $\mathcal{S}_1$ generates a random $Y \leftarrow E$ and returns $(Y_p^i, Y_u^i, replace^i) = (Y, \emptyset, 1)$. In any other case, $\mathcal{S}_1$ returns $(Y_p^i, Y_u^i, replace^i) = (\emptyset, \emptyset, 0)$. $\mathcal{S}_1$ saves its own state and the state of the servers.

The inputs in the ideal scenario are chosen uniformly at random. In the real scenario, all inputs $r_0, s_0, \ldots, s_{N-1}$ are chosen at random, while $r_1, \ldots, r_N, s_N$ are indistinguishable from random. Now, in the ith round, if the servers behave honestly, then both $\mathcal{T}$ and $\mathcal{S}_1$ correctly execute Iso, the latter choosing not to replace the output. If either server returns a wrong E_K or $\mathcal{H}_0^K$, then this will result in both $\mathcal{T}$ and $\mathcal{S}_1$ returning $\perp$. If either server returns a wrong $\mathcal{H}_1'^K$ or $\mathcal{H}_2'^K$, then both $\mathcal{T}$ and $\mathcal{S}_1$ return $\perp$ with probability at most $1-(N+1)^{-1}2^{-Nt}$. In the converse case, where the servers succeed in returning an undetected wrong output to $\mathcal{T}$, $\mathcal{S}_1$ simulates this by returning a random point on the elliptic curve. Thus, even if one of the servers behaves dishonestly in the ith round, we have $\mathcal{EVIEW}^i_{\text{real}} \sim \mathcal{EVIEW}^i_{\text{ideal}}$. It follows that $\mathcal{EVIEW}_{\text{real}} \sim \mathcal{EVIEW}_{\text{ideal}}$.

– **Pair Two:** $\mathcal{UVIEW}_{\text{real}} \sim \mathcal{UVIEW}_{\text{ideal}}$
The same PPT simulator $\mathcal{S}_2$ works for *a* secret or (honest/adversarial) protected. In round i, $\mathcal{S}_2$ generates $2(N+1)$ random scalars $u_0, \ldots, u_N, v_0, \ldots, v_N$ and makes the queries

$$(E_K, \mathcal{H}_0^K \cup \mathcal{H}_1'^K) \leftarrow \mathcal{U}_1(E, \mathcal{K}, \mathcal{H}_0 \cup \{(\{u_0, \ldots, u_N\}, Q)\}),$$
$$(E'_K, \mathcal{H}_0'^K \cup \mathcal{H}_2'^K) \leftarrow \mathcal{U}_2(E, \mathcal{K}, \mathcal{H}_0 \cup \{(\{v_0, \ldots, v_N\}, Q)\}).$$

to the servers. Then $\mathcal{S}_2$ saves its own state and the states of the servers. The inputs in the ideal scenario are chosen uniformly at random. In the real scenario, all inputs $r_0, s_0, \ldots, s_{N-1}$ are chosen at random, while $r_1, \ldots, r_N, s_N$ are indistinguishable from random to the servers. In the ith round of the real scenario, $\mathcal{T}$ always re-randomizes its inputs to the servers, while in the ideal experiment, $\mathcal{S}_2$ always creates new, random queries. Thus, for each round, we have $\mathcal{UVIEW}^i_{\text{real}} \sim \mathcal{UVIEW}^i_{\text{ideal}}$ and it follows that $\mathcal{UVIEW}_{\text{real}} \sim \mathcal{UVIEW}_{\text{ideal}}$.

The parameter *t*. In some particular scenarios, the parameter t can not only influence the verifiability and cost of the underlying system, but also its security. Related attacks become unfeasible, if the size of t reflects the security of the underlying cryptosystem against both classical and quantum attackers, i.e. in general we need to ensure that guessing all $c_1, \ldots, c_N$ correctly is at least as hard as some targeted security level 2^λ, i.e.

$$(N+1)2^{Nt} \approx 2^\lambda\,, \quad \text{or} \quad t \approx \frac{\lambda}{N}\,.$$

In this case, using the result from Section 3.2, the protocol cost per hidden point becomes

$$\mu^{-1}T_{\mathrm{OMTUP}}(\mu, \lambda/N) = (4N+3)m + \frac{2\lambda}{N}(D+A) + \left(2^{N+1} - N - 3\right)A\,.$$

By minimizing this cost with respect to N, we can find optimal values of N for different λ. This will be discussed in Section 5 in more detail.

Note that choosing $tN = \lambda$ further implies a verifiability of $1 - O(2^{-\lambda})$, which is very close to 1 for a cryptographically sized λ.

Communication versus computational cost. While algorithms under the OMTUP-assumptions will turn out to be more efficient than the algorithms under the HBC-assumption, the communication cost is generally much higher since we have to send and receive at least $2(N+1)$ elements per shrouded scalar, instead of only one. While the OMTUP case from Figure 2 has the goal of optimizing the cost, we can also easily implement the OMTUP case as two parallel invocations of the HBC case, one for each server, and compare the results. This strongly reduces communication, but increases the computational effort by the delegator to the same as in the HBC case. In general, it is possible to transform the OMTUP approach into different sets of trade-offs between these two possibilities. An easy (although not very optimal) way of doing this is to reduce N, which reduces the communication cost while increasing the computational effort by the delegator at the same time.

3.3 Difference to delegation of modular exponentiations

In delegation schemes of modular exponentiations (e.g. [14,29,32]), a delegator usually wants to outsource the computation of $(a, u) \to u^a$ for elements $u \in \mathbb{Z}_p^*$ and $a \in \mathbb{Z}_q$, for some previously defined p, q. In general, both u and a are secret or (honest/adversarial) protected. In order to shroud elements in an efficient way, the delegator has lookup-tables at its disposal, which consist of pairs $(\alpha, g^\alpha) \in \mathbb{Z}_q \times \mathbb{Z}_p^*$ that can be easily combined to obtain new pairs, e.g. $(\alpha_1, g^{\alpha_1}) \circ (\alpha_2, g^{\alpha_2}) = (\alpha_1 + \alpha_2, g^{\alpha_1} \cdot g^{\alpha_2}) = (\alpha_1 + \alpha_2, g^{\alpha_1 + \alpha_2})$. Next to being used in the shrouding step, these pairs also allow the delegator to compute part of the result itself by a single multiplication by e.g. multiplying the combination of the server's results by a final g^α to compute u^a.

We want to point out a few key differences to isogeny delegation schemes. First of all, in contrast to modular exponentiations, the domain and codomain of isogenies are different (except in the trivial case where $\mathcal{K} = \emptyset$), and more importantly, these are a priori unknown to the delegator. This means that the delegator not only has to verify if the codomain is correct, but also can not generate points on the codomain before the delegation step is completed. This also means that lookup-tables with points in the domain and codomain curves are not possible, hence the delegator can compute the final result only from linear combinations of elements the server(s) returned. Another circumstance of isogenies is that elliptic curves can not be combined in an easy way without computing isogenies,

which means that combinations, such as $(A, E_A) \circ (B, E_B) = ((A, B), E_{AB})$ are not available to the delegator, since they would defeat the purpose of delegation in the first place.

Now we turn our attention to what the delegator actually can do. One of the most important properties of isogenies in this context is that they are group homomorphisms. This means that linear combinations of points on the domain curve still hold on the codomain curve and can therefore be used to shroud and verify points, as Iso does. In order to verify the codomain curve, there seems to be no efficient way except for including at least one honest server, which will consistently return the correct curve and verify the malicious servers' results against it. The honest server is also necessary to verify if mapped points are correct. If none of the servers were honest, all points could be scaled by some previously determined factors, returning wrong results, which would still satisfy the verification conditions. In the case of unprotected points, malicious servers could even return any point instead of the desired ones.

4 Shrouding isogenies

4.1 Hiding the kernel generator

An attempt of hiding the kernel generator of a delegated isogeny was presented with the HIso algorithm of [37]. We show that this scheme is not secure and that the secret can be recovered using pairings. We then present new approaches of hiding the kernel generator of a delegated isogeny.

Pairing attack on HIso. For any given point $R = r_1P + r_2Q \in E[\tau]$ with P and Q known, 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 between a supersingular elliptic curve E and a multiplicative group G, then we can compute

$$e(R, Q) = e(r_1P + r_2Q, Q) = e(r_1P, Q) = e(P, Q)^{r_1},$$
$$e(P, R) = e(P, r_1P + r_2Q) = e(P, r_2Q) = e(P, Q)^{r_2},$$

and extract the factors r_1 and r_2 by computing the discrete logarithm using the Pohlig-Hellman algorithm [41,46], which is efficient for smooth group orders. We show that given any element $R = xA \in \langle A \rangle$ for some secret $A = a_1P + a_2Q \in E[\tau]$ with $\langle P, Q \rangle = E[\tau]$ known, we can extract an element in $[A]$. This is trivial, if $\gcd(x, \tau) = 1$, since then $R \in [A]$. If however $\gcd(x, \tau) \neq 1$, we can decompose $x = \lambda\tau'$, such that $\tau' | \tau$ and $\gcd(\lambda, \tau) = 1$. The attacker can easily compute the order τ/τ' of xA and extract τ'. It then defines $P' = \tau'P$ and $Q' = \tau'Q$ and computes $e(P', Q')$, then performs the above attack with

$$e(xA, Q') = e(P', Q')^{\lambda a_1} \quad \text{and} \quad e(P', xA) = e(P', Q')^{\lambda a_2},$$

and can construct $\lambda a_1 P + \lambda a_2 Q \in [A]$.

The concept of HIso was to delegate part of the way from e.g. E to $E/\langle A\rangle$ for a secret $A \in E[\ell^e]$, by letting the server compute a smaller isogeny in the same torsion group, $E \to E/\langle \ell^k A\rangle$ for $k = \frac{1}{3}\log_l p$, then computing the rest of the way locally. From $A_k = \ell^k A = \ell^k a_1 P + \ell^k a_2 Q$, however, the server can then extract a_1 and a_2 and reconstruct A as explained above. Hence, the scheme in [37] does not satisfy the desired security.

Shrouding isogenies via isogenies. Using the concept of shrouding isogenies via isogenies, we aim to hide the kernel generator $A \in E[\tau_A]$ via the isogenies generated by a coprime torsion group $E[\tau_K]$ with $\tau_K \approx \tau_A$. Let $K \in E[\tau_K]$ and $\hat{K} \in E[\tau_K]\backslash\langle K\rangle$. We then find the commutative diagram from Figure 3.

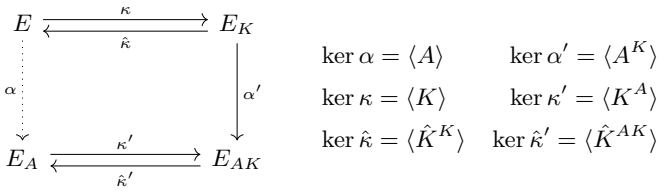


Fig. 3. Detour from $E \to E_A$ via E_K and E_{AK} and the associated kernel generators

The idea concerning the delegation is to go from E to $E/\langle A\rangle$ via the path

$$E \xrightarrow{\kappa} E_K \xrightarrow{\alpha} E_{AK} \xrightarrow{\kappa'} E_A,$$

hiding A (or α) via the isogeny $A^K = \kappa(A)$. The knowledge of $[A^K]$ does not give any information about $[A]$ by the DPP-assumption (Problem 3). Note that our approach necessarily has to take at least three steps, since any linear combination of A with elements from $E[\tau_K]$ (i.e. any "shortcut") would always reveal information about A, simply by mapping out the τ_K-torsion elements. Similarly, any shorter isogeny smaller than the length of $\tau_A \approx \tau_K$ would reduce the security of the system.

Another important aspect is that any server that has computed the delegation in Step 2 should not see any information of the delegation performed in Steps 1 or 3 (and vice versa), since the knowledge of K (or $\hat{K}^{AK}$) and A_K can be used to recover A. This is not the case for Steps 1 and 3, which can be securely delegated to the same set of servers. We therefore need to work with two sets of servers in order to be able to perform these four steps. We will denote these as $\mathbf{U}_1$ and $\mathbf{U}_2$, each being composed of one or more servers according to the underlying server assumptions (e.g. HBC, OMTUP, etc). We further note, that in the OMTUP case, the malicious servers within $\mathbf{U}_1$ and $\mathbf{U}_2$ might try to exchange their knowledge about their kernel generators indirectly. We address these issues in our delegation algorithm.

4.2 The IsoDetour-Algorithm

In this section, we present the IsoDetour-Algorithm, that uses the commutative diagram from Figure 3 in order to delegate α via a detour over the curves E_K and E_{AK}.

Definition 6 (The IsoDetour-algorithm). *The* isogeny detour delegation algorithm IsoDetour *takes as inputs a supersingular elliptic curve $E/\mathbb{F}_{p^2}$, a kernel generator $A \in E[\tau_A]$, two scalar-point pair sets $\mathcal{H}_0, \mathcal{H} \subset \mathbb{Z} \times E(\mathbb{F}_{p^2}) \backslash E[\tau_A]$, and a torsion-group indicator I. It then computes the isogeny $\phi : E \to E_A$ as $\phi = \kappa' \circ \alpha' \circ \kappa$ defined via the kernels $\ker \kappa = \langle K \rangle$, $\ker \alpha' = \langle A^K \rangle$ and $\ker \kappa' = \langle \hat{K}^{AK} \rangle$, where $K, \hat{K} \in E[\tau_I]$, both of full order and such that $\langle \hat{K} \rangle \neq \langle K \rangle$.* IsoDetour *then produces the output $(E_A; \mathcal{H}_0^A, \mathcal{H}^A)$. The inputs $E, \mathcal{H}_0$ are honest, unprotected parameters. $\mathcal{A}$ is secret, or (honest/adversarial) protected and $\mathcal{H}$ contains honest, unprotected points and secret or (honest/adversarial) protected scalars. The outputs E_A and $\mathcal{H}_0^A$ are unprotected while $\mathcal{H}^A$ is secret or protected. We write*

$$(E_A; \mathcal{H}_0^A, \mathcal{H}^A) \leftarrow \mathsf{IsoDetour}(E, A, I; \mathcal{H}_0, \mathcal{H}).$$

The implementation of this algorithm with Iso as a subroutine can be found in Figure 4. We assume that the generators $\langle P_I, Q_I \rangle = E[m_I]$ are known.

IsoDetour$(E, A, I; \mathcal{H}_0, \mathcal{H})$

1. Generate distinct $k, \hat{k} \in \mathbb{Z}_I^*$ and let $\mathcal{H}_0' = \mathcal{H}_0 \cup \{Q \mid (a, Q) \in \mathcal{H}\}$.
2. Delegate to server (group) $\mathbf{U}_1$ (in the OMTUP case, choose $tN \geq \lambda$)
$$(E_K; \{P_I, Q_I\}^K \cup \mathcal{H}_0'^K, \{A\}^K) \leftarrow \mathsf{Iso}(E, \{P_I, (k, Q_I)\}; \{P_I, Q_I\} \cup \mathcal{H}_0', \{A\}).$$
3. Delegate to server (group) $\mathbf{U}_2$ (in the OMTUP case, choose $tN \geq \lambda$)
$$(E_{AK}; \{P_I\}^{AK} \cup \mathcal{H}_0'^{AK}, \{\hat{k} Q_I\}^{AK}) \leftarrow \mathsf{Iso}(E_K, A^K; \{P_I\}^K \cup \mathcal{H}_0'^K, \{(\hat{k}, Q_I)\}^K).$$
4. Compute $\hat{K}^{AK} = P_I^{AK} + \hat{k} Q_I^{AK}$. Then extract $\mathcal{H}_0^{AK}$ and $\{Q^{AK} \mid (a, Q) \in \mathcal{H}\}$ from $\mathcal{H}_0'^{AK}$ and create $\mathcal{H}^{AK} = \{(a, Q^{AK}) \mid (a, Q) \in \mathcal{H}\}$.
5. Delegate to server (group) $\mathbf{U}_1$
$$(E_A; \mathcal{H}_0^A, \mathcal{H}^A) \leftarrow \mathsf{Iso}(E_{AK}; \hat{K}^{AK}, \mathcal{H}_0^{AK}, \mathcal{H}^{AK}).$$
6. Return $(E_A; \mathcal{H}_0^A, \mathcal{H}^A)$.

Fig. 4. Implementation of the IsoDetour algorithm given in Definition 6 using the Iso algorihm from Definition 5 as a subroutine

Mapping points. An important aspect of SIDH and related protocols (such as SIKE [2,39] and the PKE from [30]) is that there are two large torsion groups $E[\tau_A]$, $E[\tau_B]$ with generators P_A, Q_A and P_B, Q_B, respectively. Each party chooses a torsion group, in which it computes its isogeny. Then it transports the generators of the other torsion group via its isogeny to the codomain curve in order to create their public key, e.g. the public key of Alice is (E_A, P_B^A, Q_B^A). These point maps turn out to be a problem for the $\mathsf{IsoDetour}$-algorithm, since for any point $L \in E[\tau_B]$ with $L \notin \langle K \rangle$, we have $L^K \in \langle \hat{K}^K \rangle$, and thus $L^{AK} \in \langle \hat{K}^{AK} \rangle$. This implies is that any element in the τ_B-torsion group of E (or E_K) will map to $\mathcal{O}$ (or $\langle K^A \rangle$) on $E/\langle A \rangle$, and we are not able to push P_B, Q_B through this path. We present two ways to circumvent this problem below. We also note that due to the security constraints of $\mathsf{IsoDetour}$, we can also not map points in $E[\tau_A]$ to E_A. We note that the latter is also not necessary for the cryptographic protocols analyzed in this work.

More torsion groups. Assuming the protocol has more torsion groups than two, we can easily transport Bob's kernel generators $P_B, Q_B \in E[\tau_B]$ by doing a detour via isogenies defined over a third torsion group. More generally, let $p = \prod_{i=1}^{n} \tau_i \mp 1$ with $n > 2$, then Alice can delegate the computation of her public key (E_A, P_B^A, Q_B^A) via

$$(E_A; \{P_B, Q_B\}^A, \emptyset) \leftarrow \mathsf{IsoDetour}(E, A, I; \{P_B, Q_B\}, \emptyset),$$

where $I \neq A, B$.

Working with twists. We now again look at the case where we have a prime $p = \tau_A \tau_B c - 1$, where c is a small cofactor. Given $E/\mathbb{F}_{p^2}$ with $\#E = p + 1$, we find the torsion group structure

$$E(\mathbb{F}_{p^2}) \simeq (\mathbb{Z}_A)^2 \times (\mathbb{Z}_B)^2.$$

If there are only these two torsion groups at our disposal on E, we can use twists to generate "new" torsion groups [16]. To that end, we use the observation, that the quadratic twist E^t satisfies $\#E^t = p - 1$. Assuming the prime decomposition $p - 1 = \prod_{i=1}^{m} f_i^{e_i}$, we then have

$$E^t(\mathbb{F}_{p^2}) \simeq \prod_{i=1}^{m} (\mathbb{Z}/f_i^{e_i}\mathbb{Z})^2 \,.$$

Choosing a subset $S \subseteq \{1, \ldots, m\}$ of the factors $\tau_S = \prod_{i \in S} f_i^{e_i}$, such that $\tau_S \approx \tau_A \approx \tau_B$ and all coprime, we can delegate the public key computation via

$$(E_A; \{P_B, Q_B\}^A, \emptyset) \leftarrow \mathsf{IsoDetour}(E, A, S; \{P_B, Q_B\}, \emptyset),$$

by running over the twists $E \simeq E^t \to E_K^t \to E_{AK}^t \to E_A^t \simeq E_A$.

For efficiency reasons, we will have to assume that τ_S is smooth. There are not many primes p such that $p+1$ and $\tau_S \mid p-1$ are smooth. We call primes of this type *delegation-friendly primes* and generalize them in the following definition.

Definition 7 (Delegation-friendly primes). *An n-*delegation-friendly prime *is a prime p with n smooth factors $\prod_{i=1}^{n} \tau_i \mid p \pm 1$ and at least one smooth factor $\tau_S \mid p \mp 1$, such that $\tau_i \approx \tau_S$ for all i.*

A discussion surrounding delegation-friendly primes in the SIDH setting can be found in Appendix A.

The choice of t. We would like to point out the issues outlined in Remark 2.4 in [29], which in short states that "*the adversarial, unprotected input must be empty*". In the algorithm of Figure 4, the kernel generators A^K and $\hat{K}^{AK}$ actually do constitute adversarial unprotected inputs, allowing a covert channel between the two server sets, that could allow their cooperation in recovering the secret. As an example, the malicious server in $\mathbf{U}_1$ could return a wrong A^K which would covertly include information about K, allowing the extraction of A. Similarly, after the second delegation phase, $\mathbf{U}_2$ might succeed to transmit information about A^K. $\mathcal{T}$ has to mitigate this threat and can actually do so by increasing the parameter t in order to make this attack at least as hard as breaking the underlying cryptosystem (see discussion in Section 3.2). By choosing $tN \geq \lambda$ for a representative security parameter λ, $\mathcal{T}$ can guarantee that the unprotected inputs are actually honest up to a negligible probability. The same reasoning would apply to the protected outputs of the third isogeny delegation, that will determine the output of IsoDetour, yet in none of the cryptosystems analyzed in this work, protected points are required as an output.

If the outputs are not reused as adversarial unprotected inputs, $\mathcal{T}$ can choose t and N at will, only influencing the cost and verifiability of the protocol. Note that there is no advantage in choosing N different from 1 in this case.

Outsource-security of IsoDetour. We analyze the security of IsoDetour, summarized in the following theorems.

Theorem 4. *Under the honest-but-curious assumption, the outsourcing algorithm $(\mathcal{T},\mathcal{U})$ given in Figure 4 is an $\left(O\left(\frac{1}{\log\log\tau}\right),1\right)$-outsource secure implementation of* IsoDetour*, where τ is the smooth degree of the delegated isogeny.*

Theorem 5. *Under the OMTUP assumption, the outsourcing algorithm $(\mathcal{T},(\mathcal{U}_1,\mathcal{U}_2))$ given in Figure 4 is an $\left(O\left(\frac{\lambda}{\log\tau\log\log\tau}\right),1-\frac{1}{2^{t+1}}\right)$-outsource secure implementation of* IsoDetour*, where τ is the smooth degree of the delegated isogeny and λ a security parameter. If $\mathcal{H}=\emptyset$, then* IsoDetour *is even fully verifiable.*

Correctness. Correctness follows from the correctness of Iso and the commutativity of Figure 3.

Verifiability. Verifiability in the OMTUP setting also derives itself from Iso. Following the discussion from Section 4.2 and taking $t = \lambda/N$, the verifiability of the first two delegation steps is at least $1-(N+1)^{-1}2^{-\lambda}$ (see "*Verifiability*" in Section 3.2), while in the third step, the delegator can choose it individually for each point in $\mathcal{H}^{AK}$, assuming they are not later used as unprotected inputs. In order to decrease the (communication) cost, the delegator should choose $N = 1$ in the third round, yielding a verifiability of $1 - 2^{-(t+1)}$. This then also bounds the total verifiability of $\mathsf{IsoDetour}$. If $\mathcal{H}$ is empty, $\mathsf{IsoDetour}$ has a verifiability of at least $1 - (N + 1)^{-1}2^{-\lambda}$.

Cost. In total, the delegator has to delegate three times via Iso, hiding a single point in the first and in the second case, and $\mu = \#\mathcal{H}$ points in the third. Computing A^K and $\hat{K}^{AK}$ costs another two point additions. In the OMTUP case, the first two cases require $t \geq \lambda/N$, while the third one doesn't (see discussion above), and we can choose $N = 1$. Note that we assume the order of all torsion groups to be approximately τ_A. We then find the totals

$$\begin{aligned} T^{\mathrm{HBC}}_{\mathsf{IsoDet}}(\mu, \tau_A) &= 2T_{\mathrm{hbc}}(1, \tau_A) + T_{\mathrm{hbc}}(\mu, \tau_A) + 2A = (\mu + 2)S(\tau_A) + 2A\,, \\ T^{\mathrm{OMTUP}}_{\mathsf{IsoDet}}(\mu, t) &= 2T_{\mathrm{OMTUP}}(1, \lambda/N) + T_{\mathrm{OMTUP}}(\mu, t) + 2A \\ &= (8N + 6 + 5\mu)m + \left(\frac{4\lambda}{N} + 2t\mu\right) M + \left(2^{N+2} - 2N - 3 + \mu\right) A\,, \end{aligned}$$

and the associated cost reductions

$$\alpha^{\mathrm{HBC}}_{\mathsf{IsoDet}}(\mu, \tau_A) = \frac{T^{\mathrm{HBC}}_{\mathsf{IsoDet}}(\mu, \tau_A)}{I(\tau_A, \mu_0 + \mu)}\,, \quad \alpha^{\mathrm{OMTUP}}_{\mathsf{IsoDet}}(\mu, t) = \frac{T^{\mathrm{OMTUP}}_{\mathsf{IsoDet}}(\mu, t)}{I(\tau_A, \mu_0 + \mu)}\,.$$

Assuming small μ_0, μ and limiting $t \leq \lambda$, we find the following behaviors.

$$\alpha^{\mathrm{HBC}}_{\mathsf{IsoDet}}(\tau) = O\left(\frac{1}{\log\log\tau}\right)\,, \quad \alpha^{\mathrm{OMTUP}}_{\mathsf{IsoDet}}(\tau) = O\left(\frac{\lambda}{\log\tau\log\log\tau}\right)\,.$$

Security. Security of the individual steps is given by the security of the Iso algorithm. The only thing $\mathcal{T}$ has to pay close attention to, is whatever it transfers from one delegation to the next. We have shown in Section 4.2, that for the kernel generators, security is guaranteed for the appropriate choices of t and N, i.e. if $tN \geq \lambda$, where λ is the security parameter reflecting the security of the underlying cryptosystem, then we can regard the adversarial unprotected inputs (the kernel generators of rounds 2 and 3) as honest unprotected inputs, up to negligible probability. The extra data that the server sets learn (i.e. excluding the standard data in Iso) are the following:

- $\mathbf{U}_1$: $k, \hat{k}, E_K, E_{AK}$ and the generators $P_I, Q_I, P_A, Q_A P_I^K, Q_I^K, P_A^K, Q_A^K$,
- $\mathbf{U}_2$: A^K, E_K, E_{AK} and the generators $P_I^K, Q_I^K, P_I^{AK}, Q_I^{AK}$.

With the information accessible to it, $\mathbf{U}_1$ only knows the two horizontal isogenies from Figure 3. In the first round, A is hidden by the security of Iso, i.e. if $\mathbf{U}_1$ were

able to extract A, then it could be used as a subroutine to break Iso. Similarly, if after the third round, $\mathbf{U}_1$ were able to extract A from (E_K, E_{AK}), then we could trivially use it as a subroutine to break CSSI (Problem 1). Concerning $\mathbf{U}_2$, if it were able to extract $[A]$ from A^K, then we could use it as a subroutine to break DPP (Problem 3). Note that we have to pay attention not to give P_A^K, Q_A^K to $\mathbf{U}_2$ in the second round, otherwise it could recover a from A^K using the attack described in Section 4.1. If $\mathbf{U}_2$ knew P_A, Q_A and were able to compute $[P_A^K], [Q_A^K]$, then $\mathbf{U}_2$ could also be used as a subroutine to break DPP.

Concerning the mapped points in $\mathcal{H}$, the secret and protected parameters are only given to the servers in the third round. Since the torsion group generators on E_{AK} are fully verified in the HBC and OMTUP assumptions, the security of this round reduces to the security of Iso with respect to $\mathcal{H}$.

4.3 Hiding the codomain curve

Note that in some cryptographic protocols, the codomain (e.g. E_A) needs to be hidden as well. As noted in [37], the delegator needs to compute the final part of the isogeny to E_A itself, otherwise there seems to be no efficient way to hide its result from the servers. Furthermore, the size of this final isogeny would need to be at least $\tau' \approx \tau^{2/3}$ in order to yield security against a database search for E_A [37]. In this way, we can at most have a cost reduction of $I(\tau^{2/3}, n)/I(\tau, n)$, which is at least 0.6 for typically used τ.

Since the approach to HIso of [37] was broken in Section 4.1, an alternative approach using the tools developed throughout this work would need a diagram as depicted in Figure 5, i.e. a detour via four elliptic curves using two server sets to get E_A, where $\mathcal{T}$ computes the last isogeny.

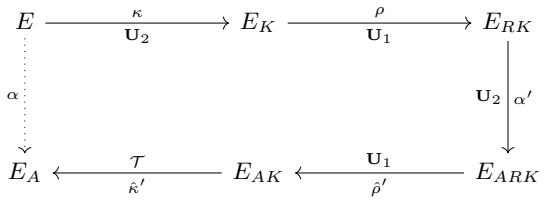


Fig. 5. An approach to hiding the codomain curve E_A via the path $E \to E_K \to E_{RK} \to E_{ARK} \to E_{AK} \to E_A$. The server groups to delegate to are also indicated in the figure. The final isogeny, $\hat{\kappa}'$ is computed by the delegator itself.

Unfortunately, with the cost of the delegation schemes themselves we get $\alpha > 85\%$. With four rounds of delegation, this approach seems unsuitable for realistic scenarios. In protocols that need a hidden codomain, we therefore assume that the delegator will need to compute them locally, for lack of a better alternative.

5 Delegation of isogeny-based protocols

In this section, we show how to apply the delegation subroutines to some of the cryptographic protocols based on supersingular isogenies over $\mathbb{F}_{p^2}$. In order to assess the computational and communication costs, we will use the 2^{e_2}-torsion groups of the standardized SIKE primes from [31].[5] Under the OMTUP-assumption, we choose the parameters $tN = e_2/2$, which reflects the classical security of the underlying protocols. The optimal value for N is then 4 for all of these primes except for $p751$, where $N = 5$ gives slightly less than a 5% when gain compared to $N = 4$. However, this comes at the cost of a higher communication, so that we will use the more realistic $N = 4$ throughout our cost assessment in this section.

To maximize efficiency, we implement the HBC case on Montgomery curves on the Kummer line. Following Remark 1, we need a group structure to implement point hiding under the OMTUP-assumption, hence we will use twisted Edwards curves in this case. While most optimized approaches in isogeny-based cryptography over $\mathbb{F}_{p^2}$ use Montgomery curves, we can easily use the transformations discussed in Section 2.2 and translate the results on twisted Edwards curves to Montgomery curves at negligible cost, allowing seamless integration of our delegation schemes into Montgomery-curve based protocols. We assume the non-delegated cases to always be performed in optimized Montgomery arithmetic.

In the following subsections, we will compare the delegated runtimes to the local (non-delegated) runtimes of some cryptographic protocols. We express our results in terms of the cost reduction function α introduced in Definition 4. To reduce communication costs we assume that servers scale points to $Z = 1$ and return them as X on Montgomery curves or $(X : Y)$ on twisted Edwards curves. In the latter case, the delegator can then easily regain T by multiplying X and Y. This increases the cost established in Section 3.2 by $(\mu_0 + \mu 2(N + 1))m$.

We present our results using the theoretical runtimes established throughout this work and compare them to benchmarks illustrating the runtimes of the delegator under both the HBC- and OMTUP-assumptions. The benchmarks were implemented using Magma v2.25-6 on an Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz with 128 GB memory. Our implementation uses parts of the Microsoft(R) `vOW4SIKE` implementation from [19].[6]

For the sake of conciseness, we assume that the protocols throughout this section are known. While we briefly review the protocol steps in order to assess the local computation cost, we refer the reader to the original sources for more details.

Remark 2 (Free Delegation). Note, that we can freely delegate any protocol that does not need hiding, i.e. where the kernel is unprotected and $\mu = 0$. Verification under the OMTUP-assumption reduces then to simple comparison operations

[5] $p434 = 2^{216}3^{137} - 1$, $p503 = 2^{250}3^{159} - 1$, $p610 = 2^{305}3^{192} - 1$, $p751 = 2^{372}3^{239} - 1$.
[6] `https://github.com/microsoft/vOW4SIKE`

and so we find a complexity of $O(1)$ for any such protocol. Some examples of such schemes are isogeny-based hash functions [12,25] with unprotected messages or verifiable delay functions [24].

5.1 Key-agreement protocols

We consider the key agreement protocols from [3,26], which are n-party extensions to SIDH [30]. In this scenario, the public parameters are the starting curve E defined over $\mathbb{F}_{p^2}$, where $p+1 = \prod_{i=1}^{n} \ell_i^{e_i}$ for n parties with different indices $i = 1, \ldots, n$, as well as kernel generators for the subgroup of each party, i.e. $\langle P_i, Q_i \rangle = E[\ell_i^{e_i}]$. The secret key of each party i is a value $a_i \in \mathbb{Z}_{\ell_i^{e_i}}$, defining $A_i = P_i + a_i Q_i$ as the kernel of $\phi_i : E \to E_i = E/\langle A_i \rangle$, while the corresponding public key contains the codomain as well as the maps of all other torsion group generators, i.e. for party i,

$$(E_i, P_1^i, Q_1^i, \ldots, P_n^i, Q_n^i)\,.$$

Public key generation step. We first show how Alice would delegate the computation of her public key. Following the concept of Section 4 and the discussion from Section 4.2, we know that she could not map Bob's kernel generators via a hidden kernel delegation along the $\ell_B^{e_B}$-torsion path. In the case $n > 2$, she can however do this via the $\ell_C^{e_C}$-torsion shrouding path. Similarly, she can map the $E[\ell_C^{e_C}]$-torsion group generators along the $\ell_B^{e_B}$-torsion shrouding path. We obtain the commutative diagram in Figure 6.

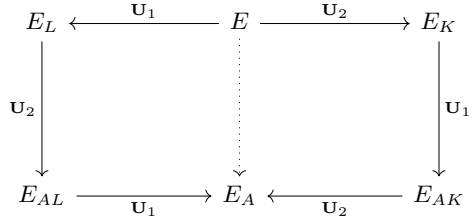


Fig. 6. Alice's concept of delegating the computation of her public key via two detours using two server groups $\mathbf{U}_1$ and $\mathbf{U}_2$.

More generally, let $I_1 \neq I_2$ be coprime torsion group indices and let Alice's group index be 1. Then Alice can delegate the computation of her public key using $\mathsf{IsoDetour}$ twice, i.e. once for each path.

$$(E_{A_1}; \mathcal{N}_1^{A_1}, \emptyset) \leftarrow \mathsf{IsoDetour}(E, A_1, I_1; \mathcal{N}_1, \emptyset),$$
$$(E_{A_1}; \mathcal{N}_2^{A_1}, \emptyset) \leftarrow \mathsf{IsoDetour}(E, A_1, I_2; \mathcal{N}_2, \emptyset),$$

where $\mathcal{N}_1 \cup \mathcal{N}_2 = \{(P_i, Q_i)\}_{i \in \{2,\ldots,n\}}$, the set of all other torsion group generators on E, such that $\mathcal{N}_1 \cap \mathcal{N}_2 = \emptyset$ and $(P_{I_1}, Q_{I_1}) \in \mathcal{N}_2$ and $(P_{I_2}, Q_{I_2}) \in \mathcal{N}_1$.

As indicated in Section 4.2, we need twice the amount of servers for an instance of IsoDetour as we would need for Iso. By using two parallel instances of IsoDetour with alternating server groups $\mathbf{U}_1$ and $\mathbf{U}_2$ as also indicated in Figure 6, we still only need two groups and three rounds.

If we have an n-delegation-friendly prime at our disposal, the public-key computation step can be delegated using a single instance of IsoDetour. Let in this case S label the smooth torsion group on the twist of the initial curve. Alice can then delegate the public key computation step via

$$(E_{A_1}; \mathcal{N}^{A_1}, \emptyset) \leftarrow \mathsf{IsoDetour}(E, A_1, S; \mathcal{N}, \emptyset)\,,$$

where $\mathcal{N} = \{(P_i, Q_i)\}_{i\in\{2,\ldots,n\}}$.

Let $d \in \{0, 1\}$ distinguish, if we have an n-delegation-friendly prime ($d = 1$) or not ($d = 0$) at our disposal. The cost reduction for public-key delegation can then be expressed as

$$\alpha_{\mathrm{PubKey},n}(d, \tau_A) = \frac{(2-d)T_{\mathsf{IsoDet}}(0, \tau_A)}{I(\tau_A, 3(n-1)) + S(\tau_A) + A}$$

In Figure 7 we depict and compare our theoretical estimate of the public-key computation with the benchmarked results for the special case $n = 2$, which is used in most standard cryptographic protocols. Note that in this case, a delegation-friendly prime is necessary. The figure shows that the gain for the delegator increases with the security level. It further shows that our theoretical predictions slightly underestimate the cost reduction via delegation. This discrepancy is mainly due to the computational overhead of local isogeny computations. The overhead becomes less important for higher degree isogenies, since the cost of isogeny computation itself increases.

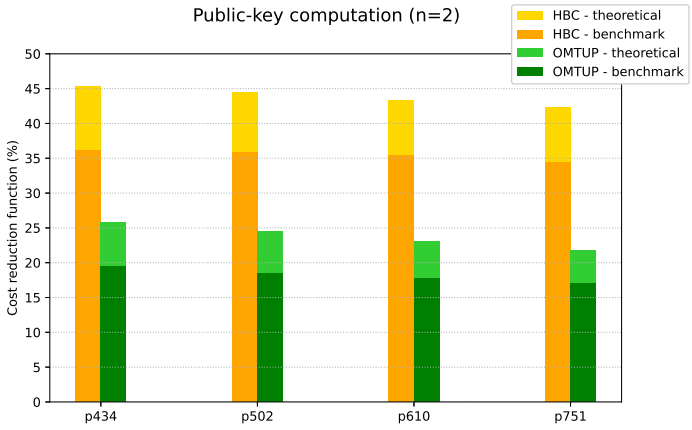


Fig. 7. Theoretical and benchmarked cost reduction function for delegating public-key computations of 2-party protocols in the HBC and OMTUP assumptions

The communication cost is given by the inputs and outputs within the three rounds of IsoDetour. We assume that the initial E and its torsion group generators are known by the servers. We note that the kernel generators in Figure 4 are computed locally and we thus have $Z \neq 1$, which increases the upload cost. The results are summarized in Table 1.

Table 1. Upload and Download costs (per server) of delegating public-key computation of 2-party protocols in the HBC and OMTUP assumptions. The results are given in kilobytes.

	$p434$		$p503$		$p610$		$p751$	
	HBC	OMT	HBC	OMT	HBC	OMT	HBC	OMT
Upload	1.30	2.83	1.50	3.28	1.82	3.98	2.25	4.90
Download	1.59	4.44	1.84	5.15	2.23	6.25	2.75	7.70

Key agreement steps. After computing her public-key, Alice still has to perform two types of steps in a n-party key agreement protocol.

Intermediate steps. In the case $n > 2$, after computing her public key, Alice has to perform $n-2$ intermediate steps. We label these with the index $k = 2, \ldots, n-1$ ($k = 1$ corresponding to the public-key generation step). At step k, Alice has to compute $(E_{k'}, \mathcal{N}^{k'})$ from $(E_k, \mathcal{N}^k \cup \{(P_A^k, Q_A^k)\})$ and her secret a_1, where $E_{k'} = E_k/\langle P_A^k + a_1 Q_A^k \rangle$ and where $\mathcal{N}^k = \{(P_i^k, Q_i^k)\}_{i \in \{k+1,\ldots,n\}}$ and $\mathcal{N}^{k'} = \{(P_i^{k'}, Q_i^{k'})\}_{i \in \{k+1,\ldots,n\}}$. Alice could delegate computations of this type using IsoDetour. However, in this scenario, it is actually cheaper to compute A_1^k locally, then delegate the isogeny using Iso

$$(E_{k'}; \mathcal{N}^{k'}, \emptyset) \leftarrow \mathsf{Iso}(E_k, A_1^k; \mathcal{N}^k, \emptyset)\,.$$

Note that A_1^k does not reveal any information about A_1 because of the difficulty of solving the Decisional Point Preimage Problem 3.

Final step. Alice's final step is the computation of the shared secret. As discussed in Section 4.3, this step needs to be computed locally. It involves the computation of the kernel generator and then of the final isogeny.

Cost. We establish the total cost of an n-party key agreement protocol. Let $d \in \{0, 1\}$ again distinguish if we have a delegation-friendly prime ($d = 1$) or not ($d = 0$) at our disposal. The public-key is computed using $2 - d$ invocations of IsoDetour with $\mu = 0$. The $n-2$ intermediate computations can then each be delegated using Iso with $\mu = 0$. The final step is then be computed locally at the cost of $S(\tau_A) + A + I(\tau_A, 0)$. Since after the public-key computation, Alice does not need to hide any points in either of the steps, she can simply perform all

of these computations on Montgomery curves, reducing her computational and communication cost. We find the total cost of

$$T_{n\mathrm{PDH}}(d, \tau_A) = (2-d)T_{\mathsf{IsoDet}}(0) + (n-1)(S(\tau_A) + A) + I(\tau_A, 0)\,,$$

under both the HBC and OMTUP assumptions.[7] In the local version of the protocol, Alice has to transport $2(n-k)$ points in round k, and compute the map of A given her generators on each curve, so that the total local cost of this protocol becomes

$$n(S(\ell_A^{e_A}) + A) + \sum_{k=1}^{n} I(\tau_A, 2(n-k)) = n(S(\ell_A^{e_A}) + A + I(\tau_A, n-1)).$$

We find

$$\alpha_{n\mathrm{PDH}}(d, \tau_A) = \frac{(2-d)T_{\mathsf{IsoDetour}}(0) + (n-1)(S(\tau_A) + A) + I(\tau_A, 0)}{n(S(\ell_A^{e_A}) + A + I(\tau_A, n-1))}.$$

We compare our theoretical estimates and benchmarks for 2-party SIDH (where $d = 1$) in Figure 8. Figure 9 further shows the evolution of the cost reduction for $p434$ in terms of n, the number of parties in the key-agreement, for both the cases with and without delegation-friendly primes.

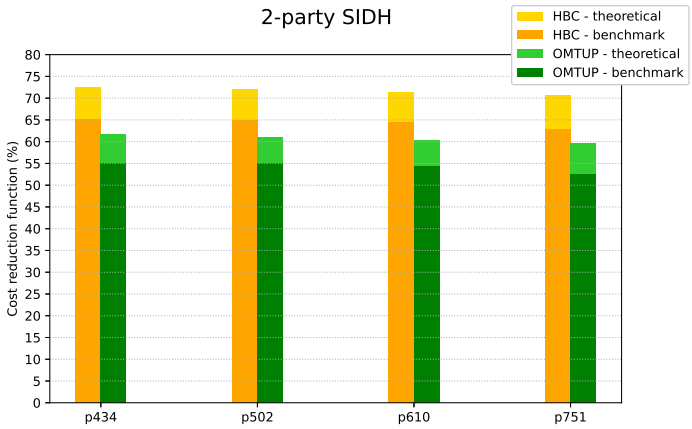


Fig. 8. Theoretical and benchmarked cost reduction function for delegating SIDH in the HBC and OMTUP assumptions

Concerning the communication costs, we also distinguish the case with and without delegation-friendly primes in Table 2 for different n. In the intermediate steps, Alice has to transport $2(n-k)$ unprotected points. Since the final step is computed locally, no communication costs apply.

[7] $T_{\mathsf{IsoDet}}(0)$ denotes a placeholder for the either $T^{\mathrm{HBC}}_{\mathsf{IsoDet}}(\mu = 0, \tau_A)$ or $T^{\mathrm{OMTUP}}_{\mathsf{IsoDet}}(\mu = 0, t)$ of Section 4.2 depending on the underlying assumption.

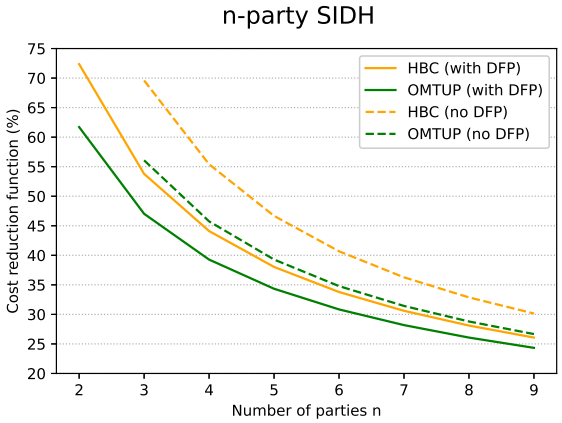


Fig. 9. Theoretical evolution of the cost reduction function for $p434$ with the number of parties in the n-party key-agreement protocol in the HBC and OMTUP assumptions. We distinguish the cases with and without a delegation-friendly prime

Table 2. Upload and Download costs (per server) of delegating the n-party key agreement protocols in the HBC and OMTUP assumptions. We distinguish the cases with and without a delegation-friendly prime. The results are given in kilobytes. Note that the communication costs for $n = 2$ (with DFP) are the same as the costs in Table 1.

		no DFP				DFP			
		$p434$		$p751$		$p434$		$p751$	
		HBC	OMT	HBC	OMT	HBC	OMT	HBC	OMT
$n = 3$	Upload	3.95	7.68	6.84	13.32	2.24	4.11	3.88	7.12
	Download	4.76	11.74	8.25	20.35	2.54	6.03	4.40	10.45
$n = 5$	Upload	7.32	12.61	12.68	21.85	4.77	7.41	8.27	12.85
	Download	8.57	18.08	14.85	31.34	5.08	9.83	8.80	17.05

Remark 3. Note that the computational and communication cost established throughout this section also apply to the delegation of isogeny-based public-key encryption [30] and key encapsulation [39] as the steps of these protocols are the same (up to some negligible computations) as (2-party) SIDH.

5.2 Identification protocols and signatures

In this section, we establish the costs of identification protocols and signature schemes. We assume the public key (E_A, P_B^A, Q_B^A) to be precomputed as it is directly related to the identity of the prover.

Zero-knowledge proof of identity. We show how the ZKPI-protocol from [30] can be delegated. In every round of the protocol, the prover needs to compute

the isogenies $\beta : E \to E_B$, $\beta' : E_A \to E_{AB}$ and the map A^B of the prover's secret. This can be done by delegating

$$(E_B; \emptyset, A^B) \leftarrow \mathsf{Iso}(E, \{P_B, (b, Q_B)\}; \emptyset, A)\,,$$
$$(E_{AB}; \emptyset, \emptyset) \leftarrow \mathsf{Iso}(E_A, \{P_B^A, (b, Q_B^A)\}; \emptyset, \emptyset)\,.$$

Depending on the challenge, the response is either b or A^B for $c = 0, 1$, respectively. If $c = 0$, the verifier delegates

$$(E_B; \emptyset, \emptyset) \leftarrow \mathsf{Iso}(E; \{P_B, (b, Q_B)\}; \emptyset, \emptyset)$$
$$(E_{AB}; \emptyset, \emptyset) \leftarrow \mathsf{Iso}(E_A; \{P_B^A, (b, Q_B^A)\}, \emptyset, \emptyset)\,,$$

otherwise $(E_{AB}; \emptyset, \emptyset) \leftarrow \mathsf{Iso}(E_B, A^B; \emptyset, \emptyset)$.

Signature schemes. We can easily extend the identification scheme to the signature schemes presented in [27]. Similarly, the delegation procedure of signature schemes is analogous to the identification schemes with the small difference that the delegator still needs to compute hash-functions. We assume that these hash functions are computed locally (or that they can be delegated with other schemes) and that their cost is negligible compared to the isogeny computations. For each of the commitments, the prover and/or verifier proceed exactly as in the identification protocol.

Cost. Following the discussion from Section 3.2, since A^B might be used as an unprotected input by the verifier, we have to choose $tN \geq \lambda$, so the cost for the prover becomes $T_{\mathrm{OMTUP}}(1, N/\lambda)$ in the OMTUP assumption. In the HBC assumption, we find $T_{\mathrm{HBC}}(1)$. For both cases, we get the cost reduction functions

$$\alpha_{\mathsf{ZKPI.P}}(\tau_B) = \frac{T(1)}{2(S(\tau_B) + A) + I(\tau_B, 1) + I(\tau_B, 0)}\,, \quad \alpha_{\mathsf{ZKPI.V}} = O(1)$$

Figure 10 shows theoretical estimates and benchmarked results for ZKPI-delegation by the prover. The theoretical predictions again underestimate the cost reduction via delegation, due to the overhead in isogeny computations. The discrepancy is higher this time higher than in Figure 7 due to the much lower cost for the delegator. Again, the gain increases with higher security.

For the communication cost, we assume that the starting curve E and the associated generators are known by the servers. In the case of the prover, we further assume that its public key E_A and associated generators are also known to the servers. We also assume that the ephemeral parameter b has to be transmitted only once to the servers. Since the OMTUP case reduces to simple comparison operations for the verifier, these can also be done on Montgomery curves, saving some of the communication. The associated upload and download costs are summarized in Table 3.

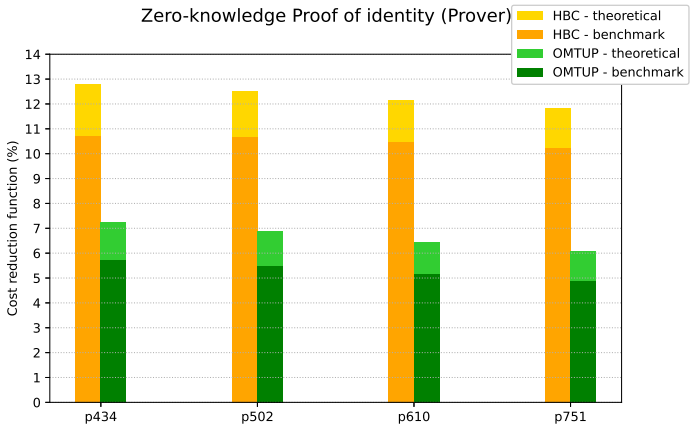


Fig. 10. Theoretical and benchmarked cost reduction function of the prover delegating zero-knowledge proofs of identity in the HBC and OMTUP assumptions

Table 3. Upload and Download costs (per server) of delegating the zero-knowledge proof of identity in the HBC and OMTUP assumptions, as well as for the verifier. The results are given in bytes. The cost for the verifier is averaged over both challenge scenarios.

	$p434$			$p751$		
	HBC	OMTUP	Ver.	HBC	OMTUP	Ver.
Upload	54	189	298	94	328	516
Download	433	1516	162	751	2628	282

6 Conclusion and future work

In this work, we presented two outsource-secure delegation schemes, Iso and IsoDetour under the *one honest-but-curious* (HBC) and *one-malicious version of a two untrusted program* (OMTUP) models of [29]. The first algorithm, Iso, allows the delegation of isogeny computations, while pushing through points hidden from the auxiliary servers. The second algorithm, IsoDetour uses Iso as a subroutine and allows to delegate the computation of an isogeny, while keeping the kernel hidden from the auxiliary servers.

We introduced the concept of delegation-friendly primes (DFP), which allow the implementation of IsoDetour in the 2-party key exchange case. For $n > 2$, this is not necessary, but increases efficiency of the delegation protocol. It is left open, if there is a more efficient way to find DFPs than the methods introduced in [16], or, more generally, if there is a way around using DFPs in the 2-party case.

Our delegation algorithms can be used as a toolbox to delegate common isogeny-based cryptographic protocols in a secure and verifiable manner. We showed how to apply Iso and IsoDetour to several isogeny-based cryptographic schemes. Our approach reduces the cost of the zero-knowledge proof of identity from [30] as well as the related signature schemes from [27] to about 11% of the original cost in the HBC case and 6% in the OMTUP case. While the cost of n-party key-exchange delegation strongly decreases with increasing n, the case $n = 2$ only reaches a reduction to about 65% of the original cost. It is of substantial interest to further reduce this number in order to make e.g. the standardization candidate SIKE efficiently delegatable. While we were able to reduce the public-key generation step in the SIDH setting to about 30% and 20% of the original cost in the HBC and OMTUP cases, respectively, the main open question in these protocols remains how to efficiently delegate the computation of an isogeny where both the kernel and codomain curve are hidden from the servers. We leave it open to apply the proposed delegation algorithms to other interesting isogeny-based schemes over $\mathbb{F}_{p^2}$. We further note that any protocol that does not need hiding of data is virtually free to delegate. Examples include hashing functions with unprotected messages [12,25] and the verifiable delay function proposed in [24].

We generally find, that while HBC has a much cheaper communication cost and is fully verifiable, our OMTUP implementations result in lower computational cost for the delegator. Further, in all the schemes of Section 5, OMTUP has a very high verifiability, close to 1. It would be interesting to see, if other server assumptions are possible in the isogeny framework, especially using only malicious servers, such as the *two-untrusted program* (TUP) or *one-untrusted program* (OUP) models introduced in [29].

For future work, it is also of interest to construct delegation algorithms for other isogeny-based schemes, such as CSIDH [11] and CSI-FiSh [8] over $\mathbb{F}_p$, or the endomorphism ring based signature protocol of [27] as well as SQI-Sign [23].

References

1. Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, John Kelsey, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, and Daniel Smith-Tone. Status report on the second round of the NIST post-quantum cryptography standardization process. *NISTIR 8309*, 07/2020 2020.
2. 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.
3. Reza Azarderakhsh, Amir Jalali, David Jao, and Vladimir Soukharev. Practical supersingular isogeny group key agreement. *IACR Cryptol. ePrint Arch.*, 2019:330, 2019.
4. Reza Azarderakhsh, Elena Bakos Lang, David Jao, and Brian Koziel. Edsidh: Supersingular isogeny Diffie-Hellman key exchange on Edwards curves. In *Inter-*

national Conference on Security, Privacy, and Applied Cryptography Engineering, pages 125–141. Springer, 2018.
5. Daniel Bernstein, Luca De Feo, Antonin Leroux, and Benjamin Smith. Faster computation of isogenies of large prime degree. *arXiv preprint arXiv:2003.10118*, 2020.
6. Daniel J Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. Twisted Edwards curves. In *International Conference on Cryptology in Africa*, pages 389–405. Springer, 2008.
7. DJ Bernstein and T Lange. Explicit-formulas database. *https://www.hyperelliptic.org/EFD*, 2019.
8. 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.
9. Cyril Bouvier and Laurent Imbert. An alternative approach for SIDH arithmetic. *IACR Cryptol. ePrint Arch.*, 2020, 2020.
10. 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.
11. 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.
12. Denis X Charles, Kristin E Lauter, and Eyal Z Goren. Cryptographic hash functions from expander graphs. *Journal of Cryptology*, 22(1):93–113, 2009.
13. Xiaofeng Chen, Jin Li, Jianfeng Ma, Qiang Tang, and Wenjing Lou. New algorithms for secure outsourcing of modular exponentiations. *IEEE Transactions on Parallel and Distributed Systems*, 25(9):2386–2396, 2014.
14. 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.
15. Andrew M. Childs, David Jao, and Vladimir Soukharev. Constructing elliptic curve isogenies in quantum subexponential time. *J. Mathematical Cryptology*, 8(1):1–29, 2014.
16. Craig Costello. B-SIDH: supersingular isogeny Diffie-Hellman using twisted torsion. *IACR Cryptol. ePrint Arch.*, 2019:1145, 2019.
17. Craig Costello and Huseyin Hisil. A simple and compact algorithm for SIDH with arbitrary degree isogenies. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 303–329. Springer, 2017.
18. Craig Costello, Patrick Longa, and Michael Naehrig. Efficient algorithms for supersingular isogeny Diffie-Hellman. In *Annual International Cryptology Conference*, pages 572–601. Springer, 2016.
19. Craig Costello, Patrick Longa, Michael Naehrig, Joost Renes, and Fernando Virdia. Improved classical cryptanalysis of the computational supersingular isogeny problem. *IACR Cryptol. ePrint Arch.*, 2019:298, 2019.
20. Craig Costello and Benjamin Smith. Montgomery curves and their arithmetic: The case of large characteristic fields. *IACR Cryptology ePrint Archive*, 2017:212, 2017.
21. Jean-marc Couveignes. Hard homogeneous spaces, 2006. https://eprint.iacr.org/2006/291.pdf.

22. Luca De Feo. Mathematics of isogeny based cryptography. *arXiv preprint arXiv:1711.04062*, 2017.
23. Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. SQISign: compact post-quantum signatures from quaternions and isogenies. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 64–93. Springer, 2020.
24. Luca De Feo, Simon Masson, Christophe Petit, and Antonio Sanso. Verifiable delay functions from supersingular isogenies and pairings. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 248–277. Springer, 2019.
25. Javad Doliskani, Geovandro CCF Pereira, and Paulo SLM Barreto. Faster cryptographic hash function from supersingular isogeny graphs. *IACR Cryptol. ePrint Arch.*, 2017:1202, 2017.
26. Satoshi Furukawa, Noboru Kunihiro, and Katsuyuki Takashima. Multi-party key exchange protocols from supersingular isogenies. In *2018 International Symposium on Information Theory and Its Applications (ISITA)*, pages 208–212. IEEE, 2018.
27. Steven D Galbraith, Christophe Petit, and Javier Silva. Identification protocols and signature schemes based on supersingular isogeny problems. *Journal of Cryptology*, 33(1):130–175, 2020.
28. Huseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, and Ed Dawson. Twisted Edwards curves revisited. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 326–343. Springer, 2008.
29. Susan Hohenberger and Anna Lysyanskaya. How to securely outsource cryptographic computations. In *Theory of Cryptography Conference*, pages 264–282. Springer, 2005.
30. 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.
31. Samuel Jaques and John M Schanck. Quantum cryptanalysis in the RAM model: Claw-finding attacks on SIKE. In *Annual International Cryptology Conference*, pages 32–61. Springer, 2019.
32. Mehmet Sabır Kiraz and Osmanbey Uzunkol. Efficient and verifiable algorithms for secure outsourcing of cryptographic computations. *International Journal of Information Security*, 15(5):519–537, 2016.
33. Michael Meyer, Steffen Reith, and Fabio Campos. On hybrid SIDH schemes using Edwards and Montgomery curve arithmetic. *IACR Cryptol. ePrint Arch.*, 2017:1213, 2017.
34. Peter L Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of computation*, 48(177):243–264, 1987.
35. 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.
36. NIST. NIST post-quantum cryptography PQC, 2020. https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.
37. 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.
38. Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

39. SIKE. Supersingular Isogeny Key Encapsulation, 2018. https://sike.org.
40. Joseph H Silverman. *The arithmetic of elliptic curves*, volume 106. Springer Science & Business Media, 2009.
41. Martin Lysoe Sommerseth and Haakon Hoeiland. Pohlig-Hellman applied in elliptic curve cryptography. 2015.
42. Anton Stolbunov. Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves. *Adv. in Math. of Comm.*, 4(2):215–235, 2010.
43. Andrew V Sutherland. Identifying supersingular elliptic curves. *LMS Journal of Computation and Mathematics*, 15:317–325, 2012.
44. 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.
45. William C Waterhouse. Abelian varieties over finite fields. In *Annales scientifiques de l'École Normale Supérieure*, volume 2, pages 521–560, 1969.
46. Nolan Winkler. The discrete log problem and elliptic curve cryptography.

A Delegation-friendly primes

Let $\tau_A = 2^{e_2}$ and $\tau_B = 3^{e_3}$ as in the SIDH setting. In order to find a 2-delegation-friendly prime (Definition 7) in this setting, we used the approach presented in [16], which uses the extended Euclidean algorithm. We first choose $a \leftarrow 2^{e_2}3^{e_3}$ and $b \leftarrow \prod_i^n \ell_i^{e_i} \approx \sqrt{a}$ coprime to a, where the ℓ_i are small primes bound by a fixed n. We then search for $s, t \in \mathbb{Z}$, such that $sa + tb = 1$ with $|s|$ small, and where $|sa - tb| = p$ is prime (for more details, cf. [16]). If this is the case, then

$$\begin{aligned} p + 1 &= 2|s|a = 2^{e_2+1}3^{e_3}|s|\,, \text{ and} \\ p - 1 &= 2|t|b = 2|t| \prod_i \ell_i^{e_i}\,, \end{aligned}$$

and we can set $\tau_A = 2^{e_2+1}$, $\tau_B = 3^{e_3}$ and $\tau_S = \prod_i \ell_i^{e_i}$

An example prime we found using this method, and representing the NIST-1 security level is the following:

$p =$ `0x48126f2641dabaf550b925fcc833262eb7c974c962aad6bf6565db634622`
`56b3468e522f111e85e2c416a82c0c5739c81af4c650000000000000000000`
`0000000000000000000000000000000000001`

which has $\tau_A = 2^{220}$, $\tau_B = 3^{147}$ and $s \approx 2^{177}$, and

$$\begin{aligned} \tau_S = 17 \cdot 29 \cdot 41 \cdot 47 \cdot 53 \cdot 59 \cdot 61 \cdot 67 \cdot 79 \cdot 103^3 \cdot 109 \cdot 113 \cdot 139 \cdot 157^2 \cdot 163 \cdot 199^2 \\ \cdot 229^2 \cdot 257^2 \cdot 311 \cdot 331 \cdot 359 \cdot 401 \cdot 457 \cdot 467\,. \end{aligned}$$

Remark 4. We would like to point out the differences to the special primes introduced in [16] and also used in [23]. While conceptually related, these primes are

defined as having two smooth torsion groups, one on each "side" of the twists, i.e. there exist smooth τ_A, τ_S, such that $\tau_A \mid p \pm 1$ and $\tau_S \mid p \mp 1$. Delegation-friendly primes on the other hand require two smooth torsion groups τ_A, τ_B on the "frontside" and at least one smooth torsion group τ_S on the "backside".