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

Robi Pedersen[1][0000−0001−5120−5709] and Osmanbey Uzunkol[2][0000−0002−5151−3848]

[1] imec-COSIC KU Leuven, Kasteelpark Arenberg 10 Bus 2452, 3001 Leuven, Belgium
robi.pedersen@esat.kuleuven.be
[2] Information und Kommunikation, Flensburg University of Applied Sciences, Flensburg, Germany
osmanbey.uzunkol@gmail.com

Abstract. Although isogeny-based cryptographic schemes enjoy the smallest key sizes amongst current post-quantum cryptographic candidates, they come at a high computational cost, making 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 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 by a factor 5 and zero-knowledge proofs of identity 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, the number of interconnected devices using new computational paradigms such as cloud, edge and mobile computing, and their interactions with 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 for researchers and industry alike. 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 [27], introduced in the context of modular exponentiations. In this model, a weak, trusted client $\mathcal{T}$ makes queries to a set of untrusted external servers $\mathcal{U}$ in such a way that their interaction $\mathcal{T}^{\mathcal{U}}$ realizes a computational task 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 $\mathcal{U}$.

Isogenies and Cryptography. Many currently deployed public-key cryptographic primitives are based on the infeasibility of either the factorization or discrete logarithm problems. Possible efficient implementations of Shor's algorithm [39] 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 [35].

One of these alternate candidates is the key encapsulation scheme SIKE [40] which is based on the Supersingular Isogeny Diffie-Hellman (SIDH) key exchange protocol, originally proposed by Jao and De Feo [29]. SIDH is a quantum resistant key agreement scheme, which uses isogenies between supersingular elliptic curves over finite fields $\mathbb{F}_{p^2}$. Besides the key agreement scheme in [29] and SIKE [40], several other cryptographic schemes based on the supersingular elliptic curves have been recently proposed in the literature ranging from group key agreement schemes [3,24], zero-knowledge proofs of identity [29,19], identification and signature schemes [25] and hash functions [11,23] to verifiable delay functions [22].

Motivation. A significant advantage of isogeny-based cryptographic schemes are the small key sizes when compared to their lattice- or code-based post-quantum counterparts. However, the main drawback is performance: SIKE is about an order of magnitude slower than its NIST competitors [1,8]. Furthermore, as pointed out in [34], 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. We address this problem 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. Recently, Pedersen and Uzunkol [36] proposed two isogeny delegation algorithms in the *honest-but-curious* (HBC) and *one-malicious version* *of a two-untrusted program* (OMTUP) assumptions using the security model of Hohenberger and Lysyanskaya [27]. 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. Random torsion point generation was done using lookup-tables of the form $\{(i, \ell^i P)\}_{i\in\{1,\dots,e-1\}}, \{(i, \ell^i Q)\}_{i\in\{1,\dots,e-1\}}$ for generators $\langle P, Q\rangle \in E[\ell^e]$. The second algorithm, HIso, used ScIso as a subroutine and allowed to hide the kernel and the codomain of the delegated isogeny. The work of [36] 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 [27] in the HBC and OMTUP models, 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 show how to break the HIso subroutine of [36] using pairings, and discuss some new approaches to hide the codomain curve in delegation algorithms.
2. We introduce the delegation algorithm Iso, which allows to delegate isogeny computations with unprotected kernel and to push through public and hidden points. Iso does not require lookup-tables, eliminating the large local memory requirement of the ScIso-algorithm from [36] on the delegator's side, while also speeding up the delegation algorithms.
3. The second algorithm, IsoDetour, uses Iso as a subroutine and allows to delegate the computation of an isogeny without revealing the kernel. This allows the computation of public keys, a question left open in [36]. The security of IsoDetour is based on a difficulty assumption implicitly used in the identification protocol of [29], which we introduce as the *decisional point preimage problem* (DPP). We show that this problem reduces to the decisional supersingular product problem (DSSP) introduced in [29].
4. We discuss applications of algorithms to the protocols introduced in [3,11,19,22,23,24,25,29] and benchmark our delegation algorithms 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. Iso allows to reduce the delegator's cost in the identification protocols of [19,29] to about 6% of the local computation cost in the OMTUP and 11% in the HBC assumption for $p503$. On the other hand, IsoDetour allows to reduce the cost of SIDH-type public-key generation to about 20% and 35% for OMTUP and HBC, respectively.

Acknowledgments. 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. This work was supported in part by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101020788 - Adv-ERC-ISOCRYPT), the Research Council KU Leuven grant C14/18/067, and by CyberSecurity Research Flanders with reference number VR20192203.

2 Background

2.1 Elliptic curves and isogenies

We work with supersingular elliptic curves over the field $\mathbb{F}_{p^2}$ with p prime and with Frobenius trace $t_\pi = \mp 2p$. The group of points on elliptic curves of this type is given as $E(\mathbb{F}_{p^2}) \simeq (\mathbb{Z}/(p \pm 1)\mathbb{Z})^2$ [41], so that the choice of p allows full control of the subgroup structure. Like most isogeny-based schemes, e.g. [29,40], we use $t_\pi = -2p$. 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 slightly abuse notation and write e.g. $P \in E$ for $P \in E(\mathbb{F}_{p^2})$. We indicate by $E[\tau]$ 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 shorthand $\mathbb{Z}_\tau = \mathbb{Z}/\tau\mathbb{Z}$. We assume that different torsion groups are always coprime.

Isogenies. Isogenies are homomorphisms between two elliptic curves, that are also algebraic maps [18,41]. 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 $A \in E[\tau_A]$, i.e. $\langle A \rangle = \{\lambda A | \lambda \in \mathbb{Z}_{\tau_A}\} \subset E[\tau_A]$, 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 $[A]$ of points generating $\langle A \rangle$. 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. In order to allow efficient isogeny computations between elliptic curves, torsion groups $E[\tau]$ need τ to be smooth [29]. 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.

Notation. We write isogeny codomains 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 isogeny kernel generator. We 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. $\{P, Q\}^A = \{P^A, Q^A\} \subset E_A$.

2.2 Elliptic curve arithmetic

Computational costs. We denote by A and D the theoretical cost estimates of point addition and point doubling on E, respectively, by $\mathsf{S}(\tau)$ the cost estimate of a (large) scalar multiplication of a point by a scalar in $\mathbb{Z}_\tau$ and by $\mathsf{I}(\tau,\mu)$ the cost estimate of computing a (large) τ-isogeny, and pushing μ points through this isogeny. Each of these operations can be expressed 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 $0.8\mathsf{m}$, while addition on $\mathbb{F}_{p^2}$ and comparisons are negligible. Expensive inversions are circumvented by using projective coordinates. Large scalar multiplications are typically done using a double-and-add approach, so that we can express the cost of scalar multiplication by an element τ as [36]

$$\mathsf{S}(\tau) = \mathsf{M}\lceil \log_2 \tau \rceil - \mathsf{A}\,, \qquad \text{where} \qquad \mathsf{M} = \mathsf{A} + \mathsf{D}\,. \tag{1}$$

Scalar multiplications by a small prime ℓ_i are written as S_{ℓ_i}. We further define C_{ℓ_i} and P_{ℓ_i} as the cost of a computing the codomain of an ℓ_i-isogeny and evaluating an ℓ_i-isogeny respectively. In Appendix A, we establish the following cost of a τ-isogeny with $\tau = \prod_{i=1}^n \ell_i^{e_i}$:

$$\mathsf{I}(\tau,\mu) = \sum_{i=1}^{n} \Big[(\mathsf{P}_{\ell_i} + \mathsf{S}_{\ell_i}) \frac{e_i}{2} \log_2 e_i + (\mathsf{C}_i + \mu \mathsf{P}_i) e_i \Big] + \sum_{i=1}^{n-1} \mathsf{P}_{\ell_i} e_i (n-i)\,. \tag{2}$$

We will work with elliptic curves in Montgomery and in twisted Edwards form with extended coordinates. Due to their more efficient arithmetic, we assume that isogeny computations are always performed using the former.

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 $a^2 \neq 4$. Montgomery curves are used in most deployed isogeny-based protocol, as arithmetic operations and isogeny computations on them 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 [17,33]. 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 [16]. 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 [14].

Using the results from [6,17], the cost of point addition and doubling on $E(\mathbb{F}_{p^2})$ can be estimated by

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

respectively. Scalar multiplications on Montgomery curves are performed using the Montgomery ladder algorithm [33]. Finally, using the optimized results from [15] for the parameters in the isogeny computation (2), we find

$$\begin{aligned} \mathsf{C}_3 &= 4.4m, \quad \mathsf{S}_3 = 9.2m, \quad \mathsf{P}_3 = 5.6m, \\ \mathsf{C}_4 &= 3.2m, \quad \mathsf{S}_4 = 7.2m, \quad \mathsf{P}_4 = 7.6m. \end{aligned}$$

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 [5], the extended coordinates introduced in [26] are of particular interest for this work. In [26], point coordinates are extended by a fourth element $T = XY/Z$, which allows for efficient arithmetic with

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

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 $\mathsf{A} = 8m$. 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 [5], 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 [5,9,32]

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

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.

2.3 Security model

The security model for delegating cryptographic computations used throughout this paper was originally proposed by Hohenberger and Lysyanskaya [27]. 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 ($\mathcal{U}$ can be seen to be installed behind $\mathcal{T}$'s firewall). 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 [27]), while the non-secret outputs do not contain any sensitive data. The security of delegation schemes is formalized in the following definition, which also formalizes $\mathcal{T}$'s efficiency gain due to the delegation, as well as its ability to verify correctness of $\mathcal{U}$'s outputs.

Definition 1 ((α, β)-outsource-security [27]). *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. We write $\mathcal{E}VIEW_{real} \sim \mathcal{E}VIEW_{ideal}$ ($\mathcal{E}$ learns nothing) and $\mathcal{U}VIEW_{real} \sim \mathcal{U}VIEW_{ideal}$ ($\mathcal{U}$ learns nothing). The details of these experiments can be found in Definition 2.2 of [27]. If $\mathcal{U}$ consists of multiple servers, then there is a PPT-simulator $\mathcal{S}_{2,i}$ for each of their views.*
– ***Cost reduction:*** *for all inputs x, the running time of $\mathcal{T}$ is at most an α-multiplicative factor of the running time of $\mathsf{Alg}(x)$,*
– ***Verifiability:*** *for all inputs x, if $\mathcal{U}$ deviates from its advertised functionality during the execution $\mathcal{T}^{\mathcal{U}}(x)$, then $\mathcal{T}$ will detect the error with probability $\geq \beta$.*

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 [12]). *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 [27]). *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 [27] 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 Appendix B.1.

2.4 Cryptographic protocols and difficulty assumptions

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

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

Let $\langle A \rangle = \ker \alpha$ and $\langle B \rangle = \ker \beta$, then the commutativity of the upper diagram is given by choosing $\ker \alpha' = \langle A^B \rangle$ and $\ker \beta' = \langle B^A \rangle$.

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 [29].

Problem 1 (Computational Supersingular Isogeny Problem (CSSI) [29]). 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) [29]). 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 random with $|E_1| = |E|$ 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)). Given (E, E_B, A, A'^B), where $A \in E[\tau_A]$, and $A'^B \in E_B[\tau_A]$, decide whether $[A] = [A']$.

Let $\mathcal{A}_{\text{DPP}}$ be an adversary to the DPP problem which, upon receiving the tuple (E, E_B, A, A'^B), returns $b = 1$ if $[A^B] = [A'^B]$, otherwise $b = 0$. Then, we can construct an adversary $\mathcal{B}^{\mathcal{A}_{\text{DPP}}}_{\text{DSSP}}$ against DSSP, which returns $b = 0$ in the first and $b = 1$ in the second case of Problem 2. Upon receiving $(E, E_A, E_B, E_C, \alpha, \alpha')$, $\mathcal{B}^{\mathcal{A}_{\text{DPP}}}_{\text{DSSP}}$ 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 $\mathcal{A}_{\text{DPP}}$. $\mathcal{B}^{\mathcal{A}_{\text{DPP}}}_{\text{DSSP}}$ returns what $\mathcal{A}_{\text{DPP}}$ returns: 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 $E/\mathbb{F}_{p^2}$ be an elliptic curve, $\mathcal{K} \subset \mathbb{Z}_\tau \times E[\tau]$, $\mathcal{M} \subset \mathbb{Z} \times E$ two distinct sets of scalar-point pairs and $b \in \{0, 1\}$ a bit. We assume that the delegator gives inputs of the form $(E, \mathcal{K}; \mathcal{M}; b)$ to the server, who proceeds as follows.

- $\mathcal{K}$ encodes the kernel of the isogeny to compute, thus the server computes $K = \sum_{(a,P)\in\mathcal{K}} aP$, which it uses to compute the isogeny $\phi : E \to E_K$. Throughout this work, we are only interested in sets of the form $\mathcal{K} = \{(1, P), (k, Q)\}$ for generators $\langle P, Q\rangle = E[\tau]$.
- $\mathcal{M}$ contains points to push through and multiply with the associated scalar, i.e. the server computes $\mathcal{M}^K := \{aX^K \mid (a, X) \in \mathcal{M}\}$, where $X^K = \phi(X)$.
- If $b = 1$, the server generates a deterministic *return basis* $\mathcal{B}^K = \{R^K, S^K\} \subset E_K[\tau]$, such that $R^K + kS^K = P^K$.[3] If $b = 0$, then $\mathcal{B}^K = \emptyset$.

The server then returns $(E_K; \mathcal{M}^K; \mathcal{B}^K)$. We write the delegation step as follows

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

The points in $\mathcal{M}$ are always submitted in a random order in order to avoid distinguishability. Further, to reduce the communication cost we assume that servers return all points scaled with $Z = 1$.

Notation. For a scalar-point pair (a, P) in $\mathcal{K}$ or $\mathcal{M}$, we simply write P if $a = 1$. If a set contains multiple pairs of the same point, e.g. $\{(a_1, P), (a_2, P), (a_3, P)\}$, we condense them as $\{(\{a_1, a_2, a_3\}, P)\}$.

[3] This can simply be achieved by first generating $S^K \in E_K[\tau]$ deterministically (e.g. by hashing into the elliptic curve using a procedure such as the one described in [28], and map out the unwanted torsion), then computing $R^K = P^K - kS^K$.

3.2 The Iso-Algorithm

Definition 4 (The Iso-algorithm). *The* isogeny delegation algorithm Iso *takes as inputs a supersingular elliptic curve $E/\mathbb{F}_{p^2}$, a kernel set $\mathcal{K} \subset \mathbb{Z} \times E(\mathbb{F}_{p^2})$, two scalar-point pair sets $\mathcal{H}_0, \mathcal{H} \subset \mathbb{Z} \times E(\mathbb{F}_{p^2})$ and a bit $b \in \{0,1\}$, then computes the isogeny $\phi : E \to E_K$ and produces the output $(E_K; \mathcal{H}_0^K, \mathcal{H}^K; \mathcal{B}^K)$, where $K = \sum_{(a,P)\in\mathcal{K}} aP$, $\mathcal{H}_{(0)}^K = \{aP^K \mid (a,P) \in \mathcal{H}_{(0)}\}$ and $\mathcal{B}^K$ is a return basis as described in Section 3.1, if $b = 1$ and $\emptyset$ otherwise. The inputs $E, \mathcal{K}, \mathcal{H}_0, b$ are all honest, unprotected parameters, while $\mathcal{H}$ contains secret or (honest/adversarial) protected scalars and honest, unprotected points. The outputs E_K, $\mathcal{H}_0^K$ and $\mathcal{B}^K$ are unprotected while $\mathcal{H}^K$ is secret or protected. We write*

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

If $b = 0$ and thus $\mathcal{B}^K = \emptyset$, we shorten this as $(E_K; \mathcal{H}_0^K, \mathcal{H}^K) \leftarrow \mathsf{Iso}(E, \mathcal{K}; \mathcal{H}_0, \mathcal{H})$.

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: First, auxiliary elements are generated (Gen), which are used to shroud protected elements (Shr), before being delegated to the server (Del). After the delegation, the server outputs are verified (Ver) and finally the results are recovered and output (Out). In Appendix B.1, we discuss the differences of delegations of isogenies and modular exponentiations.

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. 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. In Appendix C, we establish the protocol execution costs for the delegator

$$T_{\mathrm{HBC}}(\mu, \tau_A) = \mu \mathsf{S}(\tau_A)\,, \tag{3}$$

$$T_{\mathrm{OMTUP}}(\mu, t) = \mu \left[(4N+3)\mathsf{m} + 2\mathsf{M}t + (2^{N+1} - N - 3)\mathsf{A} \right]\,, \tag{4}$$

of Figures 1 and 2 and further prove the following theorems.

Theorem 1. *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.*

Theorem 2. *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.*

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

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

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.

Honest-but-curious approach.

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

Fig. 1. Implementation of Iso in the HBC assumption

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 $c_1,\dots,c_N,d_1,\dots,d_N\in\{-2^{t-1},\dots,2^{t-1}\}$, and
- random scalars $r_0,s_0,s_1,\dots,s_{N-1}\in\mathbb{Z}_\tau$.

Shr: For each $(a,Q)\in\mathcal{H}$, compute $r_i=-s_i+c_is_0+d_ir_0$ for $i=1,\dots,N-1$. Define $\sigma=\sum_{i=1}^{N-1}(s_i+r_i)$ and let γ be the smallest integer >1 coprime to τ, then compute $s_N=\gamma^{-1}(d_Nr_0+c_Ns_0+\sigma-a)$ and $r_N=-s_N+c_Ns_0+d_Nr_0$. Set

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

Del: Delegate $(E_K;\mathcal{H}_0^K\cup\mathcal{H}_1'^K;\mathcal{B}^K)\leftarrow\mathcal{U}_1(E,\mathcal{K};\mathcal{H}_0\cup\mathcal{H}_1';b)$ and $(E_K';\mathcal{H}_0'^K\cup\mathcal{H}_2'^K;\mathcal{B}'^K)\leftarrow\mathcal{U}_2(E,\mathcal{K};\mathcal{H}_0\cup\mathcal{H}_2';b)$.
Ver: Verify, if $E_K\stackrel{?}{=}E_K'$, $\mathcal{H}_0^K\stackrel{?}{=}\mathcal{H}_0'^K$, $\mathcal{B}^K\stackrel{?}{=}\mathcal{B}'^K$, and if $(s_iQ)^K+(r_iQ)^K\stackrel{?}{=}c_i(s_0Q)^K+d_i(r_0Q)^K$, for $i=1,\dots,N$.
Out: If any of the verifications fail, return $\perp$, otherwise return $(E_K;\mathcal{H}_0^K,\mathcal{H}^K;\mathcal{B}^K)$, where

$$\mathcal{H}^K=\left\{r_NQ^K-(\gamma-1)s_NQ^K+\sum_{i=1}^{N-1}(s_iQ^K+r_iQ^K)\middle|(a,Q)\in\mathcal{H}\right\}\,.$$

Fig. 2. Implementation of Iso in the OMTUP assumption

The parameter t. In some cases, the parameter t does 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. $(N+1)2^{Nt} \approx 2^\lambda$ or $t \approx \frac{\lambda}{N}$. In this case, using equation (4), the protocol cost per hidden point becomes

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

In Section 5, we minimize this cost with respect to N for specific choices of λ. 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.

4 Shrouding isogenies

We aim to hide the kernel generator $A \in E[\tau_A]$ via the isogenies generated by a coprime torsion group $E[\tau_I]$ with $\tau_I \approx \tau_A$. The idea is to go from E to E_A via the path $E \xrightarrow{\kappa} E_K \xrightarrow{\alpha} E_{AK} \xrightarrow{\hat{\kappa}'} E_A$, where $\hat{\kappa}'$ is the dual of κ pushed through α. The path is depicted in Figure 3. The point A (or the isogeny α) is hidden via the isogeny $A^K = \kappa(A)$, since 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 $E[\tau_I]$ (i.e. any "shortcut") would always reveal information about A by mapping out the τ_I-torsion elements. Similarly, any shorter isogeny, smaller than the length of $\tau_A \approx \tau_I$, would reduce the security of the system.

$$\begin{tikzcd}
E \arrow[r, shift left, "\kappa"] \arrow[d, dotted, "\alpha"'] & E_K \arrow[l, shift left, "\hat{\kappa}"] \arrow[d, "\alpha'"] \\
E_A \arrow[r, shift left, "\kappa'"] & E_{AK} \arrow[l, shift left, "\hat{\kappa}'"]
\end{tikzcd}$$

$$\begin{array}{ll}
\ker\alpha = \langle A \rangle & \ker\alpha' = \langle A^K \rangle \\
\ker\kappa = \langle K \rangle & \ker\kappa' = \langle K^A \rangle \\
\ker\hat{\kappa} = \langle \hat{K}^K \rangle & \ker\hat{\kappa}' = \langle \hat{K}^{AK} \rangle
\end{array}$$

Fig. 3. Detour from $E \to E_A$ via E_K and E_{AK} and the associated kernel generators. The point $\hat{K}$ is any point of full order in $E[\tau_I] \backslash \langle K \rangle$.

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. We therefore in general need to work with at multiple sets of servers, each being composed of one or more servers according to the underlying server assumptions (e.g. HBC, OMTUP). We denote these sets as $\mathbf{U}_1, \mathbf{U}_2, \mathbf{U}_3$, for delegation steps 1, 2 and 3. Under certain conditions, we can choose $\mathbf{U}_1 = \mathbf{U}_3$, which we will discuss further below, as well as in Appendix C.3. We also note, that in the OMTUP case, the malicious servers within these sets might try to exchange their knowledge about the kernel generators indirectly, which also needs to be addressed in our algorithm.

Definition 5 (The IsoDetour-algorithm). *The* isogeny detour delegation algorithm IsoDetour *takes as inputs a supersingular elliptic curve $E/\mathbb{F}_{p^2}$, a kernel generator $A = P_A + aQ_A$ where $\langle P_A, Q_A\rangle = E[\tau_A]$, two scalar-point pair sets $\mathcal{H}_0, \mathcal{H} \subset \mathbb{Z} \times E \backslash (E[\tau_A] \cup E[\tau_I])$, and a torsion-group indicator I. It then computes the isogeny $\phi : E \to E_A$ as $\phi = \kappa' \circ \alpha' \circ \kappa$ 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*

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

In Figure 4, 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}. We assume that the generators $\langle P_I, Q_I\rangle = E[\tau_I]$ are known.

IsoDetour proceeds as follows: First, the isogeny κ is delegated to $\mathbf{U}_1$ and the point A is pushed through, hidden from the servers. The servers are also prompted to return a basis $R^K, S^K \in E_K$, such that $R^K + kS^K = P^K \in \ker \hat{\kappa}$. These points will later be used to compute the "return" isogeny $\hat{\kappa}'$. The point A^K is then used as the kernel generator for α', computed by $\mathbf{U}_2$, with R^K, S^K are pushed through. Finally, the delegator constructs the kernel generator $R^{AK} + kS^{AK}$ of $\hat{\kappa}'$ for the third delegation by $\mathbf{U}_3$. For any other scalar-point pair, that we want to push through, the general idea is to extract the (unprotected) points in $\mathcal{H}_0$ and $\mathcal{H}$ and simply push them through the first two rounds of delegation; the desired multiplication with hidden scalars needs to be done in the third round only. Note that since these points are pushed through κ and later through $\hat{\kappa}'$, the result will be multiplied by a factor $\deg \kappa = \deg \hat{\kappa}' = \tau_I$. Thus, we need to multiply the related scalars with τ_I^{-1}, in order to compensate for this.

Mapping points. Note that since $\hat{\kappa}'$ is represents the dual isogeny of κ pushed through α', any points mapped via the detour path will necessarily by multiplied by τ_I. This is corrected in step 4 by multiplying these points with the inverse of τ_I. Note that this multiplication is only defined for points in torsion groups of order coprime to τ_I,[4] thus not for points in $E[\tau_I]$. An important aspect of SIDH and related protocols (such as SIKE [2,40] and the PKE from [29]) 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 IsoDetour-algorithm, since any point in $E[\tau_B]$ will map to $\mathcal{O}$ on E_A, and we are not able to map P_B, Q_B along this path. We present two ways to circumvent this

[4] We assume τ_I^{-1} to be known with respect any other torsion group.

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

  1. Generate random $k \in \mathbb{Z}^*_{\tau_I}$ and let $\mathcal{H}'_0 = \{Q \mid (a, Q) \in \mathcal{H}_0 \cup \mathcal{H}\}$.
  2. Delegate to server (group) $\mathbf{U}_1$ (in the OMTUP case, choose $tN \geq \lambda$)
         $(E_K; \mathcal{H}'^K_0 \cup \{P^K_A\}, \{aQ^K_A\}; \{R^K, S^K\}) \leftarrow \mathsf{Iso}(E, \{P_I, (k, Q_I)\}; \mathcal{H}'_0 \cup \{P_A\}, \{(a, Q_A)\}; 1)$.
     and compute $A^K = P^K_A + aQ^K_A$.
  3. Delegate to server (group) $\mathbf{U}_2$ (in the OMTUP case, choose $tN \geq \lambda$)
         $(E_{AK}; \{R^{AK}\} \cup \mathcal{H}'^{AK}_0, \{kS^{AK}\}; \emptyset) \leftarrow \mathsf{Iso}(E_K, \{A^K\}; \{R^K\} \cup \mathcal{H}'^K_0, \{(k, S^K)\}; 0)$.
  4. From $\mathcal{H}'^{AK}_0$, build $\mathcal{H}''^{AK}_{(0)} = \{(a\tau_I^{-1}, Q^{AK}) \mid (a, Q) \in \mathcal{H}_{(0)}\}$. Then, compute the kernel
     $\hat{K}^{AK} = R^{AK} + kS^{AK}$.
  5. Delegate $(E_A; \mathcal{H}^A_0, \mathcal{H}^A; \emptyset) \leftarrow \mathsf{Iso}(E_{AK}, \{\hat{K}^{AK}\}; \mathcal{H}''^{AK}_0, \mathcal{H}''^{AK}; 0)$ to server (group) $\mathbf{U}_1$.
  6. Return $(E_A; \mathcal{H}^A_0, \mathcal{H}^A)$.
```

Fig. 4. Implementation of the $\mathsf{IsoDetour}$ algorithm given in Definition 5 using the Iso algorithm from Definition 4 as a subroutine.

problem below. We also note that due to the security constraints of $\mathsf{IsoDetour}$, we also cannot map points in $E[\tau_A]$ to E_A. Fortunately, this is 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 $I \neq A, B$. 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^A_B, Q^A_B) as

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

Working with twists. If we are working with a prime of the form $p \pm 1 = f\tau_A\tau_B$, i.e. we only have two torsion groups at our disposal on E, we can use twists to generate "new" torsion groups [14] on E^t. Assuming the prime decomposition $p \mp 1 = D\tau_S$, with $\tau_S \approx \tau_A$ smooth and D a co-factor, we have another torsion group on the "backside" of our elliptic curve, $E^t[\tau_S]$. We can simply 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 $E \simeq E^t \to E^t_K \to E^t_{AK} \to E^t_A \simeq E_A$. For efficiency reasons, τ_S has to be smooth. There are not many primes p such that $p \pm 1$ and $\tau_S \mid p \mp 1$ are smooth. We call primes of this type *delegation-friendly primes* and generalize them in the following definition. We present an approach to generate such primes in Appendix B.4.

Definition 6 (Delegation-friendly primes). *An n-delegation-friendly prime (DFP) 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.*

We discuss under which conditions we can choose $\mathbf{U}_1 = \mathbf{U}_3$. An important consequence of using multiple torsion groups or delegation-friendly primes are the susceptibility to torsion-attacks as described in [37,38]. The security of such a delegation depends strongly on the points revealed on E_K and E_{AK}, which in turn reveal the action of α' on these subgroups. As an example, consider standard SIDH with a DFP, i.e. where we have $p \pm 1 = f\tau_A\tau_B$ and $p \mp 1 = \tau_S D$. Using $\mathsf{IsoDetour}$ in order to compute a public key reveals the action of α' on $E[\tau_B]$ and $E[\tau_S]$, which would allow a quadratic speedup of the isogeny recovery attack by [38, Prop. 25 and Prop.27]. In this case, we would need three sets of servers in order to not allow this attack. Taking the non-DFP $p \pm 1 = f\tau_A\tau_B\tau_I$ instead, results in a slightly less than quadratic speedup, but in more expensive arithmetic. While small speedups might in some situations not pose a problem, we will discuss in Section 5 and Appendix C.3 under which conditions these occur. Note that this does not make our schemes insecure, as we simply point out, under which conditions two server groups can be used instead of three. In the case of three different server sets, these attacks do not apply.

Choosing t. We point out the issues outlined in Remark 2.4 of [27], which in short states that "*the adversarial, unprotected input must be empty*". In Figure 4, the kernel generators A^K and $\hat{K}^{AK}$ actually do constitute adversarial unprotected inputs, and might allow the malicious server in $\mathbf{U}_1$ to communicate information about K to $\mathbf{U}_2$, revealing information about A. To mitigate this threat, $\mathcal{T}$ can increase the parameter t so far to make this attack at least as hard as breaking the underlying cryptosystem. As discussed in Section 3.2, choosing $tN \geq \lambda$ guarantees that the unprotected inputs are actually honest up to a negligible probability. Note that if such points do not constitute adversarial unprotected inputs, t and N will only influence the cost and verifiability of the protocol. There is no advantage in choosing N different from 1 in this case.

Outsource-security of $\mathsf{IsoDetour}$. In Appendix C, we derive the costs

$$T^{\text{HBC}}_{\mathsf{IsoDet}}(\mu, \tau_A) = (\mu + 2)\mathsf{S}(\tau_A) + 2\mathsf{A}\,,$$
$$T^{\text{OMTUP}}_{\mathsf{IsoDet}}(\mu, t) = (8N + 6 + 5\mu)\mathsf{m} + \left(\frac{4\lambda}{N} + 2t\mu\right)\mathsf{M} + \left(2^{N+2} - 2N - 3 + \mu\right)\mathsf{A}\,.$$

for the delegator and prove the following theorems.

Theorem 3. *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* $\mathsf{IsoDetour}$*, where* τ *is the smooth degree of the delegated isogeny.*

Theorem 4. *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* $\mathsf{IsoDetour}$*, where* τ *is the smooth degree of the delegated isogeny and* λ *a security parameter. If* $\mathcal{H} = \emptyset$*, then* $\mathsf{IsoDetour}$ *is fully verifiable.*

Hiding the kernel generator. A first attempt of hiding the kernel generator of a delegated isogeny was presented with the HIso algorithm of [36]. In Appendix B.2, we show that this scheme is not secure and that the secret can be recovered using pairings. In Appendix B.3, we discuss how this is possible using the approach presented in this section. Unfortunately, it turns out to be too expensive for realistic scenarios. In protocols that need a hidden codomain, we therefore assume that the delegator computes them locally.

5 Delegation of isogeny-based protocols

We apply our proposed 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 [30].[5] To maximize efficiency, we implement the HBC case on Montgomery curves on the Kummer line, while we need a group structure to implement point hiding under the OMTUP-assumption, hence we will use twisted Edwards curves in this case (see Remark 5). The efficient transformations between these curves allow seamless integration of our delegation schemes into typically Montgomery-curve based protocols. We assume local computations to always be performed in optimized Montgomery arithmetic.

In the following subsections, we compare the delegated runtimes to the local (non-delegated) cost of some cryptographic protocols. We express our results in terms of the cost reduction function α introduced in Definition 1. To avoid adversarial inputs in the OMTUP-assumption, we use $\lambda = e_2/2$, which reflects the classical security of the underlying protocols. The optimal value of N for all SIKE primes is $N = 4$ (also considering communication costs).

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.[6] 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 [13].[7] The necessary communication costs can be found in Appendix D.

For the sake of conciseness, we assume that the protocols in 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 1 (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 of the server outputs then reduce to simple comparison operations under the

[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] Our implementation can be found at `https://github.com/gemeis/SIDHdelegation` and includes representative benchmarks for the delegator's operations as well as a proof-of-concept implementation for the correctness of our algorithms.

[7] `https://github.com/microsoft/vOW4SIKE`

OMTUP-assumption. Some examples of such schemes are isogeny-based hash functions [11,23] with unprotected messages or verifiable delay functions [22].

5.1 Key-agreement protocols

We consider the key agreement protocols from [3,24], which are n-party extensions to SIDH [20]. In this scenario, we have $p + 1 = \prod_{i=1}^{n} \ell_i^{e_i}$ for n parties. Each party $\mathcal{P}_i$ is assigned a subgroup $\langle P_i, Q_i \rangle = E[\ell_i^{e_i}]$ and has a secret key $a_i \in \mathbb{Z}_{\ell_i^{e_i}}$, 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 is $(E_i, P_1^i, Q_1^i, \ldots, P_n^i, Q_n^i)$ for party i. While we consider the n-party case in order to stay general, we point out that n-party key agreement protocols have to be used with caution, as torsion point attacks can be quite effective in these settings. In particular, [38] presents improved attacks for $n > 2$ and a polynomial-time break for $n \geq 6$.

Public key generation step. Let Alice be $\mathcal{P}_1$. If $n > 2$, Alice can delegate her public key computation using $\mathsf{IsoDetour}$ twice, along two paths $I_1 \neq I_2$:

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

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 $(P_{I_2}, Q_{I_2}) \in \mathcal{N}_1$. By using alternating server groups $\mathbf{U}_1$ and $\mathbf{U}_2$ as indicated in Figure 5, and by carefully choosing $\mathcal{N}_1$ and $\mathcal{N}_2$, we can assure that the servers get as little information as possible about the action of the isogenies α_1' and α_2' on the torsion groups, so that we only need two server groups for delegation.[8]

$$\begin{tikzcd}
E_L \arrow[d, "\mathbf{U}_2"', "\alpha_1'"] & E \arrow[l, "\mathbf{U}_1"'] \arrow[r, "\mathbf{U}_2"] \arrow[d, dotted] & E_K \arrow[d, "\alpha_2'"', "\mathbf{U}_1"] \\
E_{AL} \arrow[r, "\mathbf{U}_1"'] & E_A & E_{AK} \arrow[l, "\mathbf{U}_2"]
\end{tikzcd}$$

Fig. 5. Alice's concept of delegating the computation of her public key via two detours using two server groups $\mathbf{U}_1$ and $\mathbf{U}_2$. L and K are from different torsion groups.

With an n-DFP, this step can be delegated with a single instance of $\mathsf{IsoDetour}$ using the smooth torsion group on the twist side. This case needs three server groups. Let $d \in \{0, 1\}$ distinguish, if we have an n-DFP ($d = 1$) or not ($d = 0$) at our disposal. The cost reduction for public-key delegation can then be expressed

[8] For example, we could simply split up generators P_i, Q_i into both sets for all i.

as

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

Figure 6 compares our theoretical estimates with the benchmarked results for $n = 2$, used in most cryptographic protocols. In this case, a delegation-friendly prime is necessary. The communication costs are summarized in Table 1.

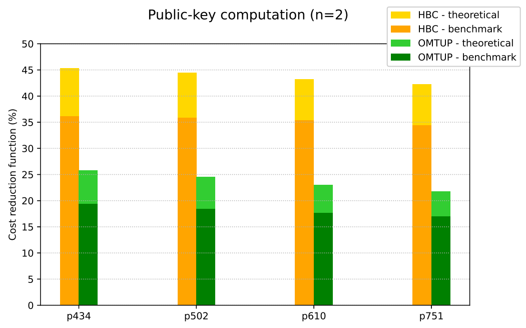

Fig. 6. Theoretical and benchmarked cost reduction function for delegating public-key computations of 2-party protocols in the HBC and OMTUP assumptions. The discrepancy between theoretical and benchmarked cost reduction is mainly due to the computational overhead of local isogeny computations. The overhead becomes less important for higher degree isogenies, since the cost of isogeny computation itself increases. We also see that the gain for the delegator increases with the security level.

Intermediate steps. If $n > 2$, Alice performs $n-2$ intermediate steps $k \in \{2,\ldots,n-1\}$, in which she has to compute $(E_{k'}, \mathcal{N}^{k'})$ from $(E_k, \mathcal{N}^k \cup \{(P_A^k, Q_A^k)\})$, where $E_{k'} = E_k/\langle P_A^k + a_1 Q_A^k\rangle$ and $\mathcal{N}^{k^{(')}} = \{(P_i^{k^{(')}}, Q_i^{k^{(')}})\}_{i\in\{k+1,\ldots,n\}}$. Note that in this scenario, it is cheaper to compute A_1^k locally and delegate

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

than using $\mathsf{IsoDetour}$. Note again 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 Sections 4 and B.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 $\mathsf{IsoDetour}$ with $\mu = 0$. The $n-2$ intermediate computations can then each be delegated using Iso with $\mu = 0$. The final step is then computed locally at the cost of $\mathsf{S}(\tau_A) + \mathsf{A} + \mathsf{I}(\tau_A, 0)$. 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

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

under both the HBC and OMTUP assumptions.[9] 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 except the first. We find

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

Figure 7 shows the evolution of the cost reduction for $p434$ in terms of n for the cases with and without delegation-friendly primes and compares our theoretical estimates and benchmarks for the 2-party case ($d = 1$). Table 1 summarizes the communication costs for different n.

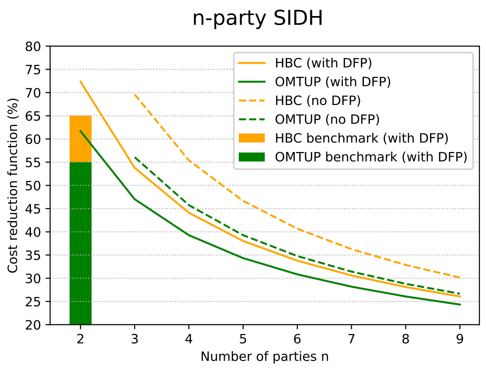

Fig. 7. Theoretical cost reduction for delegating n-party key agreement protocols for $p434$ for different n with and without a delegation-friendly prime. The case $n = 2$ further includes benchmarks.

Remark 2. Note that the computational and communication cost established throughout this section also apply to the delegation of isogeny-based public-key encryption [20] and key encapsulation [40] 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

[9] $T_{\mathsf{IsoDet}}(0)$ denotes a placeholder for either $T^{\text{HBC}}_{\mathsf{IsoDet}}(\mu = 0, \tau_A)$ or $T^{\text{OMTUP}}_{\mathsf{IsoDet}}(\mu = 0, t)$ of Section 4 depending on the underlying assumption.

directly related to the identity of the prover.

Zero-knowledge proof of identity. We show how the ZKPI-protocol from [20] 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

$$\begin{aligned}(E_B; P_A^B, aQ_A^B) &\leftarrow \mathsf{Iso}(E, \{P_B, (b, Q_B)\}; \{P_A\}, \{(a, Q_A)\})\,,\\ (E_{AB}; \emptyset, \emptyset) &\leftarrow \mathsf{Iso}(E_A, \{P_B^A, (b, Q_B^A)\}; \emptyset, \emptyset).\end{aligned}$$

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

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

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

Signature schemes. The delegation procedure of the signature schemes in [25] based on this identification scheme is completely analogous, i.e. for each of the commitments, the prover and/or verifier proceed exactly as in the identification protocol. The delegator further needs to compute hash-functions, but we assume that these have negligible cost (or are delegated with other schemes).

Remark 3. We note that an alternative ID protocol to [20] has recently been proposed in [19]. This scheme is quite similar, except that an $E_B[\tau_A]$ basis needs to be deterministically generated using an algorithm called $\mathsf{CanonicalBasis}$. We can delegate this newer scheme in exactly the same fashion as the one presented here, except that we have to add the execution of $\mathsf{CanonicalBasis}$ to the advertised server functionality. Since the algorithm is deterministic, we only have to compare the output of both servers in the OMTUP assumption, in order to verify that the output is correct. Note that the download communication cost is increased by these extra points.

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 $T_{\text{OMTUP}}(1, N/\lambda)$ in the OMTUP and $T_{\text{HBC}}(1, \tau_A)$ in the HBC assumption. For both cases, we get the cost reduction functions

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

Figure 8 shows theoretical estimates and benchmarked results for ZKPI-delegation by the prover. We summarize the communication costs in Table 2.

6 Conclusion and future work

In this work, we presented two outsource-secure delegation schemes, Iso and $\mathsf{IsoDetour}$, under the *one honest-but-curious* (HBC) and *one-malicious version*

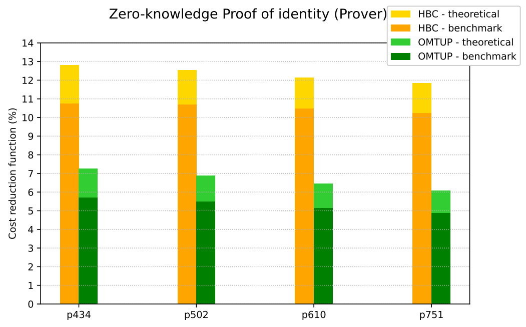

Fig. 8. Theoretical and benchmarked cost reduction function of the prover delegating zero-knowledge proofs of identity in the HBC and OMTUP assumptions. The theoretical predictions again underestimate the cost reduction via delegation, due to the overhead in isogeny computations. The discrepancy is higher this time higher than in Figure 6 due to the much lower cost for the delegator. Again, the gain increases with higher security.

of a two untrusted program (OMTUP) models of [27]. Our delegation algorithms can be used as a toolbox to delegate common isogeny-based cryptographic protocols in a secure and verifiable manner. Our approach reduces the cost of the zero-knowledge proof of identity from [29] as well as the related signature schemes from [25] 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 35% 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 [11,23] and the verifiable delay function proposed in [22].

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 [27].

For future work, it is also of interest to construct delegation algorithms for other isogeny-based schemes, such as CSIDH [10] and CSI-FiSh [7] over $\mathbb{F}_p$, or the endomorphism ring based signature protocol of [25] as well as SQI-Sign [21].

References

1. Alagic, G., Alperin-Sheriff, J., Apon, D., Cooper, D., Dang, Q., Kelsey, J., Liu, Y.K., Miller, C., Moody, D., Peralta, R., Perlner, R., Robinson, A., Smith-Tone, D.: Status report on the second round of the NIST post-quantum cryptography standardization process. NISTIR 8309 (07/2020 2020). https://doi.org/https://doi.org/10.6028/NIST.IR.8309
2. Azarderakhsh, R., Campagna, M., Costello, C., De Feo, L., Hess, B., Jalali, A., Jao, D., Koziel, B., LaMacchia, B., Longa, P., et al.: Supersingular isogeny key encapsulation. Submission to the NIST Post-Quantum Standardization project (2017)
3. Azarderakhsh, R., Jalali, A., Jao, D., Soukharev, V.: Practical supersingular isogeny group key agreement. IACR Cryptol. ePrint Arch. **2019**, 330 (2019)
4. Azarderakhsh, R., Lang, E.B., Jao, D., Koziel, B.: Edsidh: Supersingular isogeny Diffie-Hellman key exchange on Edwards curves. In: International Conference on Security, Privacy, and Applied Cryptography Engineering. pp. 125–141. Springer (2018)
5. Bernstein, D.J., Birkner, P., Joye, M., Lange, T., Peters, C.: Twisted Edwards curves. In: International Conference on Cryptology in Africa. pp. 389–405. Springer (2008)
6. Bernstein, D., Lange, T.: Explicit-formulas database. `https://www.hyperelliptic.org/EFD` (accessed 5th May 2021)
7. Beullens, W., Kleinjung, T., Vercauteren, F.: CSI-FiSh: Efficient isogeny based signatures through class group computations. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 227–247. Springer (2019)
8. Bouvier, C., Imbert, L.: An alternative approach for SIDH arithmetic. IACR Cryptol. ePrint Arch. **2020** (2020)
9. Castryck, W., Galbraith, S.D., Farashahi, R.R.: Efficient arithmetic on elliptic curves using a mixed Edwards-Montgomery representation. IACR Cryptol. ePrint Arch. **2008**, 218 (2008)
10. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: an efficient post-quantum commutative group action. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 395–427. Springer (2018)
11. Charles, D.X., Lauter, K.E., Goren, E.Z.: Cryptographic hash functions from expander graphs. Journal of Cryptology **22**(1), 93–113 (2009)
12. Chevalier, C., Laguillaumie, F., Vergnaud, D.: Privately outsourcing exponentiation to a single server: cryptanalysis and optimal constructions. In: European Symposium on Research in Computer Security. pp. 261–278. Springer (2016)
13. Costello, C., Longa, P., Naehrig, M., Renes, J., Virdia, F.: Improved classical cryptanalysis of sike in practice. In: Kiayias, A.(ed.), Public-Key Cryptography–PKC 2020: 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4–7, 2020, Proceedings, Part II. pp. 505–534. Cham: Springer International Publishing (2020)
14. Costello, C.: B-SIDH: Supersingular isogeny Diffie-Hellman using twisted torsion. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 440–463. Springer (2020)
15. Costello, C., Hisil, H.: A simple and compact algorithm for SIDH with arbitrary degree isogenies. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 303–329. Springer (2017)

16. Costello, C., Longa, P., Naehrig, M.: Efficient algorithms for supersingular isogeny Diffie-Hellman. In: Annual International Cryptology Conference. pp. 572–601. Springer (2016)
17. Costello, C., Smith, B.: Montgomery curves and their arithmetic. Journal of Cryptographic Engineering **8**(3), 227–240 (2018)
18. De Feo, L.: Mathematics of isogeny based cryptography. arXiv preprint arXiv:1711.04062 (2017)
19. De Feo, L., Dobson, S., Galbraith, S., Zobernig, L.: SIDH proof of knowledge. IACR Cryptol. ePrint Arch. **2021**, 1023 (2021)
20. De Feo, L., Jao, D., Plût, J.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. Journal of Mathematical Cryptology **8**(3), 209–247 (2014)
21. De Feo, L., Kohel, D., Leroux, A., Petit, C., Wesolowski, B.: SQISign: compact post-quantum signatures from quaternions and isogenies. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 64–93. Springer (2020)
22. De Feo, L., Masson, S., Petit, C., Sanso, A.: Verifiable delay functions from supersingular isogenies and pairings. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 248–277. Springer (2019)
23. Doliskani, J., Pereira, G.C., Barreto, P.S.: Faster cryptographic hash function from supersingular isogeny graphs. IACR Cryptol. ePrint Arch. **2017**, 1202 (2017)
24. Furukawa, S., Kunihiro, N., Takashima, K.: Multi-party key exchange protocols from supersingular isogenies. In: 2018 International Symposium on Information Theory and Its Applications (ISITA). pp. 208–212. IEEE (2018)
25. Galbraith, S.D., Petit, C., Silva, J.: Identification protocols and signature schemes based on supersingular isogeny problems. Journal of Cryptology **33**(1), 130–175 (2020)
26. Hisil, H., Wong, K.K.H., Carter, G., Dawson, E.: Twisted Edwards curves revisited. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 326–343. Springer (2008)
27. Hohenberger, S., Lysyanskaya, A.: How to securely outsource cryptographic computations. In: Theory of Cryptography Conference. pp. 264–282. Springer (2005)
28. Icart, T.: How to hash into elliptic curves. In: Annual International Cryptology Conference. pp. 303–316. Springer (2009)
29. Jao, D., De Feo, L.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: International Workshop on Post-Quantum Cryptography. pp. 19–34. Springer (2011)
30. Jaques, S., Schanck, J.M.: Quantum cryptanalysis in the RAM model: Claw-finding attacks on SIKE. In: Annual International Cryptology Conference. pp. 32–61. Springer (2019)
31. Kiraz, M.S., Uzunkol, O.: Efficient and verifiable algorithms for secure outsourcing of cryptographic computations. International Journal of Information Security **15**(5), 519–537 (2016)
32. Meyer, M., Reith, S., Campos, F.: On hybrid SIDH schemes using Edwards and Montgomery curve arithmetic. IACR Cryptol. ePrint Arch. **2017**, 1213 (2017)
33. Montgomery, P.L.: Speeding the Pollard and elliptic curve methods of factorization. Mathematics of computation **48**(177), 243–264 (1987)
34. 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

35. NIST: NIST post-quantum cryptography PQC (2020), https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions
36. Pedersen, R., Uzunkol, O.: Secure delegation of isogeny computations and cryptographic applications. In: Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop. pp. 29–42 (2019)
37. Petit, C.: Faster algorithms for isogeny problems using torsion point images. In: Advances in Cryptology – ASIACRYPT 2017. pp. 330–353 (2017)
38. de Quehen, V., Kutas, P., Leonardi, C., Martindale, C., Panny, L., Petit, C., Stange, K.E.: Improved torsion point attacks on sidh variants. arXiv preprint arXiv:2005.14681 (2020)
39. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: Proceedings of the 35th Annual Symposium on Foundations of Computer Science. pp. 124–134 (1994)
40. SIKE: Supersingular Isogeny Key Encapsulation (2018), https://sike.org
41. Silverman, J.H.: The arithmetic of elliptic curves, vol. 106. Springer Science & Business Media (2009)

A Isogeny computation cost

We establish equation (2). 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 [20]. 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 $i = 0, \dots, e-1$, 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, $\phi = \phi_{e-1} \circ \dots \circ \phi_0 : E \to E_R$. 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 [20], 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 $E_1, \dots, E_e$ 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

$$\mathsf{I}(\ell^e, \mu) = (\mathsf{P}_\ell + \mathsf{S}_\ell)\frac{e}{2}\log_2 e + (\mathsf{C}_\ell + \mu \mathsf{P}_l)e\,. \tag{5}$$

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 $i = 1, \dots, n$. 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 $i = 1, \dots, n-1$. Finally, we find the cost of a τ-isogeny from equation (2):

$$\mathsf{I}(\tau, \mu) = \sum_{i=1}^{n} \left[(\mathsf{P}_{\ell_i} + \mathsf{S}_{\ell_i})\frac{e_i}{2}\log_2 e_i + (\mathsf{C}_i + \mu \mathsf{P}_i)e_i \right] + \sum_{i=1}^{n-1} \mathsf{P}_{\ell_i} e_i (n-i)\,.$$

B Further discussions

B.1 Difference to delegation of modular exponentiations

We want to point out a few key differences of isogeny delegation schemes to those of modular exponentiation as in [12,27,31]. 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.

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.

B.2 Breaking HIso from [36]

The concept of the HIso-algorithm from [36] 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 an element from $[A]$ as follows: 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 $e(P', Q')$ and

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

From these equations, the attacker can extract λa_1 and λa_2 using the polynomial-time Pohlig-Hellman algorithm for smooth order groups. Using this, the attacker can construct $\lambda a_1 P + \lambda a_2 Q \in [A]$, hence the scheme in [36] does not satisfy the desired security.

B.3 Hiding the codomain with IsoDetour

Note that in some cryptographic protocols, the codomain (e.g. E_A) needs to be hidden as well. As noted in [36], 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 [36]. In this way, we can at most have a cost reduction of $\mathsf{I}(\tau^{2/3}, n)/\mathsf{I}(\tau, n)$, which is at least 0.6 for cryptographically sized τ.

Since the approach to HIso of [36] was broken in Section B.2, an alternative approach using the tools developed throughout this work would need a diagram as depicted in Figure 9, i.e. a detour via four elliptic curves using two server sets to get E_A, where $\mathcal{T}$ computes the last isogeny.

$$\begin{tikzcd}
E \arrow[r, "\kappa"] \arrow[d, dotted, "\alpha"'] & E_K \arrow[r, "\rho"] & E_{RK} \arrow[d, "\alpha'"] \\
E_A & E_{AK} \arrow[l, "\hat{\kappa}'"] & E_{ARK} \arrow[l, "\hat{\rho}'"]
\end{tikzcd}$$

Fig. 9. 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 final isogeny, $\hat{\kappa}'$ needs to be 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.

B.4 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 6) in this setting, we use the approach presented in [14], 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. [14]). If this is the case, then

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

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:

$$\begin{aligned}p = {} & \texttt{0x48126f2641dabaf550b925fcc833262eb7c974c962aad6bf6565db634622}\\ & \texttt{56b3468e522f111e85e2c416a82c0c5739c81af4c650000000000000000000}\\ & \texttt{000000000000000000000000000000000001}\end{aligned}$$

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 [14] and also used in [21]. 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 $\tau_S \mid p \mp 1$. 2-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”.

It is left open, if there is a more efficient way to find DFPs than the methods introduced in [14], or, more generally, if there is a way around using DFPs in the 2-party case.

C Proof of outsource-security theorems

C.1 Proof of Theorem 1

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 $\mathsf{S}(\tau_A)$, 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 $T_{\mathrm{HBC}}(\mu, \tau_A) = \mu\mathsf{S}(\tau_A)$. 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 $\mathsf{I}(\tau_K, \mu_0 + \mu)$, so that we find

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

Assuming μ small and $\tau_A \approx \tau_K$,[10] we find

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

[10] 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.

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 $\mathcal{EVIEW}_{\text{real}} \sim \mathcal{EVIEW}_{\text{ideal}}$ and $\mathcal{UVIEW}_{\text{real}} \sim \mathcal{UVIEW}_{\text{ideal}}$.

C.2 Proof of Theorem 2

Correctness. The elements $E_K, \mathcal{H}_0^K$ and $\mathcal{B}^K$ are correct due to the direct comparison between the servers (one of which is honest). Concerning $\mathcal{H}^K$, 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 wrong $E_K, \mathcal{H}_0^K$ and $\mathcal{B}^K$, 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 $2\mathsf{m}$ per r_i and another $3\mathsf{m}$ for s_N, thus $2N\mathsf{m}+3\mathsf{m}$, assuming γ^{-1} is known on the underlying field. In the verification step, the delegator has to compute $2N\mathsf{A}$ 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 $2N\mathsf{m}$.

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

Naively, the scalar multiplications in the verification step would cost $2N\mathsf{S}(2^t)$, 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 $c_1Q, \dots, c_NQ$ for the scalars $c_i = (c_{i,0} \dots c_{i,t-1})_2$ 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 $S = \mathcal{P}(\{1, \dots, N\})$, excluding the empty set. For each $k \in S$, let

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

enumerate all possible distinct areas in the Venn diagram of those sets. Then, after having computed all the doubling operations $2Q, \dots, 2^{t-1}Q$, 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 $|\bigcup_k A_k| \le t$ and $A_k \cap A_{k'} = \emptyset\ \forall k \ne k'$, 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

$$S_N(t) = \mathsf{M}t + (2^N - 2N - 2)\mathsf{A}$$

for N parallel scalar multiplications c_iQ, where $\mathsf{M} = \mathsf{D} + \mathsf{A}$.

Remark 5. 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. On the Kummer line, the delegator could not even compute e.g. $s_iQ + r_iQ$, using differential addition, since it would be lacking the knowledge of $(s_i - r_i)Q$, which it can't get from one of the servers without revealing information.

We can finally express[11]

$$\begin{aligned} T_{\text{OMTUP}}(\mu, t) &= \mu\left[(4N+3)\mathsf{m} + 3N\mathsf{A} + \mathsf{A} + 2\mathsf{S}_N(2^t)\right] \\ &= \mu\left[(4N+3)\mathsf{m} + 2\mathsf{M}t + (2^{N+1} - N - 3)\mathsf{A}\right] . \end{aligned}$$

The cost reduction can then be expressed as follows

$$\alpha_{\text{OMTUP}}(\mu_0, \mu, t, \tau) = \frac{T_{\text{OMTUP}}(\mu, t)}{\mathsf{I}(\tau, \mu_0 + \mu)} = \frac{\mu\left[(4N+3)\mathsf{m} + 2\mathsf{M}t + (2^{N+1} - N - 3)\mathsf{A}\right]}{\sum_i (\mathsf{I}_{\ell_i} + \mathsf{S}_{\ell_i} + (\mu_0 + \mu)\mathsf{P}_{\ell_i})\frac{e_i}{2}\log_2 e_i}$$

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

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

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 $Q \in E[\tau]$.

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

– **Pair One:** $\mathcal{EVIEW}_{\text{real}} \sim \mathcal{EVIEW}_{\text{ideal}}$
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 $u_0, \dots, u_N, v_0, \dots, v_N$ and makes the queries:

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

to the servers, then verifies if all the outputs are correct. If either E_K, E'_K, $\mathcal{H}_0, \mathcal{H}'_0$ or $\mathcal{B}^K, \mathcal{B}'^K$ are incorrect, $\mathcal{S}_1$ returns $(Y_p^i, Y_u^i, \mathit{replace}^i) = (\perp, \emptyset, 1)$. 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 $(Y_p^i, Y_u^i, \mathit{replace}^i) = (Y, \emptyset, 1)$. In any other case, $\mathcal{S}_1$ returns $(Y_p^i, Y_u^i, \mathit{replace}^i) = (\emptyset, \emptyset, 0)$. $\mathcal{S}_1$ 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 $r_0, s_0, \dots, s_{N-1}$ are chosen at random, while $r_1, \dots, r_N, s_N$ 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, $\mathcal{H}_0^K$ or $\mathcal{B}^K$, 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 $u_0, \dots, u_N, v_0, \dots, v_N$ and makes the queries

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

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 $r_0, s_0, \dots, s_{N-1}$ are chosen at random, while $r_1, \dots, r_N, s_N$ 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 $\mathcal{UVIEW}_{\text{real}} \sim \mathcal{UVIEW}_{\text{ideal}}$.

C.3 Proof of Theorems 3 and 4

Correctness. Correctness follows from the correctness of Iso and the commutativity of Figure 3. Correctness of the point maps follows from the discussion "Mapping points" below Figure 4.

Verifiability. Verifiability in the OMTUP setting also derives itself from Iso. Following the discussion about choosing t from Section 4, we take $t = \lambda/N$, so that the verifiability of the first two delegation steps is at least $1-(N+1)^{-1}2^{-\lambda}$, 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

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

and the associated cost reductions

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

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, 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, E_0, E_K and R, S, R^K, S^K and P_A, Q_A, P_A^K, Q_A^K,
- $\mathbf{U}_2$: A^K, E_K, E_{AK} and the generators R^K, S^K, R^{AK}, S^{AK}.
- $\mathbf{U}_3$: k, E_{AK}, E_A and the generators R^{AK}, S^{AK}

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, $\mathbf{U}_3$ 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 B.2. 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 $\mathcal{H}$.

We also consider the case with two server groups. In this case, the extra data that the server sets learn are:

- $\mathbf{U}_1$: k, E_0, E_K, E_{AK}, E_A and the generators $R, S, R^K, S^K, R^{AK}, S^{AK}$ and P_A, Q_A, P_A^K, Q_A^K,
- $\mathbf{U}_2$: A^K, E_K, E_{AK} and the generators R^K, S^K, R^{AK}, S^{AK}.

The same arguments as before apply, except that now, $\mathbf{U}_1$ knows the two horizontal isogenies from Figure 3. Using the knowledge of R^K, S^K, R^{AK}, S^{AK}, the servers $\mathbf{U}_1$ further know the action of α' on $E[\tau_I]$. Under specific circumstances, this might be a threat, as $\mathbf{U}_1$ could apply the attacks described in [38]. This attack also strongly depends on the points transported within $\mathcal{H}_0$ and $\mathcal{H}$.

In order to stay general, assume that $p = \prod_{i=1}^{n} \tau_i \pm 1$ and that $\tau_A \approx \tau_i \approx p^{1/n}$ for all i. Using the points within $E[\tau_I]$ and $\mathcal{H}_{(0)}$, we assume that the attacker knows the action of α' on m torsion groups. We collectively denote these as $\tau_B \approx p^{m/n}$. Finally, let $\tau_{B'} \approx p^{1-(m+1)/n}$ be the torsion groups, on which the action of α' is unknown to the attacker. In the case where p is a DFP, we have $\tau_{B'} \approx p^{1-m/n}$. We analyze the conditions in which the algorithms presented in [38] run faster than standard meet-in-the-middle attacks.

For $n \geq 2$, this is the case, if $\tau_B > \sqrt{p}\tau_A$ classically or $\tau_B > \sqrt{p}$ quantumly by [38, Corollary 26 and 28], which implies, $m > \frac{n+2}{2}$ classically and $m > \frac{n}{2}$ quantumly, for both DFPs and non-DFPs. If these conditions are fulfilled, the delegator therefore needs three server sets instead of two.

D Communication costs

In order to express 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. Montgomery curves are fully defined by a single parameter $a \in \mathbb{F}_{p^2}$ and points on the Kummer line can be expressed by two coordinates $X, Z \in \mathbb{F}_{p^2}$. 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.

We summarize the communication cost of n-party protocols and of the ZKPI in Tables 1 and 2, respectively.

Table 1. Upload and Download costs (in kB 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 cost is given by the inputs and outputs within the three rounds of IsoDetour, assuming the initial E and its torsion group generators are known by the servers. We note that the kernel generator $\hat{K}^{AK}$ in Figure 4 is computed locally and we thus have $Z \neq 1$, which increases the upload cost. In the intermediate steps, Alice has to transport $2(n-k)$ unprotected points. Since the final step is computed locally, no communication costs apply. Therefore, the communication for $n = 2$ is the same as the communication needed to delegate the public key computation.

		no DFP				DFP			
		$p434$		$p751$		$p434$		$p751$	
		HBC	OMT	HBC	OMT	HBC	OMT	HBC	OMT
$n = 2$	Upload	–	–	–	–	1.30	2.83	2.25	4.90
	Download	–	–	–	–	1.80	4.86	3.12	8.43
$n = 3$	Upload	3.95	7.68	6.84	13.32	2.24	4.11	3.88	7.12
	Download	5.18	12.58	8.98	21.81	2.75	6.45	4.77	11.18
$n = 4$	Upload	5.53	10.02	9.58	17.37	3.40	5.64	5.89	9.78
	Download	6.98	15.65	12.1	27.13	3.91	8.25	6.78	14.3

Table 2. Upload and Download costs (in B per server) of delegating the zero-knowledge proof of identity in the HBC and OMTUP assumptions, as well as for the verifier. The cost for the verifier is averaged over both challenge scenarios. 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. 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.

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