



# New proof systems and an OPRF from CSIDH

Cyprien Delpéch de Saint Guilhem  and Robi Pedersen 

COSIC, KU Leuven  
Kasteelpark Arenberg 10 Bus 2452, 3001 Leuven, Belgium

**Abstract.** Isogeny computations in CSIDH (Asiacrypt 2018) are described using a commutative group  $\mathcal{G}$  acting on the set of supersingular elliptic curves. The commutativity property gives CSIDH enough flexibility to allow the creation of many cryptographic primitives and protocols. Nevertheless, these operations are limited and more complex applications have not yet been proposed.

When calling the composition of two group elements of  $\mathcal{G}$  *addition*, our goal in this work is to explore exponentiation, multiplication with public elements, and multiplication between secret elements of this group. We first introduce a two-party interactive protocol for multiplication of secret group elements. Then, we explore zero-knowledge proofs of these different arithmetic operations. We present two types of approaches, using either standard sigma protocols or the MPC-in-the-Head paradigm. Most of our proofs need a trusted setup, which can be removed in the MPC-in-the-Head setting using cut-and-choose techniques. We conclude this work by presenting an oblivious pseudorandom function based on our new framework, that is competitive with current state-of-the-art designs.

**Keywords:** Isogeny-based cryptography · CSIDH · Zero-knowledge proofs · MPC-in-the-Head · Cryptographic Protocols · OPRF

## 1 Introduction

Isogeny-based assumptions are one of a few promising candidates for quantum-resistant cryptography. Even after the recent break of SIKE and SIDH [21, 52, 59] and related schemes, protocols based on different security assumptions, such as CSIDH [22] and SQISign [29], remain viable alternatives for isogeny-based constructions. The former has shown a lot of versatility over the past few years and many protocols have been based on its flexibility, such as signatures [13, 28, 33] and public-key encryption [37, 54], oblivious transfer [6, 51], oblivious pseudorandom functions [17, 42], distributed protocols [4, 5, 13–15, 23, 26, 30] and many more [1, 2, 32].

CSIDH-based schemes can be described via a group  $\mathcal{G}$  acting on a set  $\mathcal{E}$  as

$$\star : \mathcal{G} \times \mathcal{E} \rightarrow \mathcal{E}.$$

Given  $(E, \mathbf{a} \star E) \in \mathcal{E}^2$ , where  $\mathbf{a} \in \mathcal{G}$ , it is computationally infeasible to learn  $\mathbf{a}$ . Here,  $\mathcal{E}$  is the set of supersingular elliptic curves over a prime field  $\mathbb{F}_p$  with some

specified endomorphism ring. There is no defined operation on the elements of  $\mathcal{E}$ . The only flexibility allowing the construction of protocols comes from  $\mathcal{G}$ , the ideal-class group  $\text{cl}(\mathcal{O})$  of an order  $\mathcal{O}$  in  $\mathbb{Q}(\sqrt{-p})$ , which is commutative, among other properties. This allows to easily build a Diffie–Hellman scheme between two parties, with secret-public key pairs  $(\mathbf{a}, \mathbf{a} \star E)$  and  $(\mathbf{b}, \mathbf{b} \star E)$ , respectively, by each using the other’s public key to compute  $\mathbf{a} \star (\mathbf{b} \star E) = \mathbf{ab} \star E = \mathbf{b} \star (\mathbf{a} \star E)$ . This is exactly the operation underlying the Diffie–Hellman key exchange scheme from Castryck et al. [22].

Expressed in terms of a public generator  $\langle \mathbf{g} \rangle \subseteq \mathcal{G}$ , let  $\mathbf{a} = \mathbf{g}^a$  and  $\mathbf{b} = \mathbf{g}^b$ , then this Diffie–Hellman scheme amounts to the *addition* of elements in the exponent of  $\mathbf{g}$ , i.e.  $\mathbf{g}^a \star (\mathbf{g}^b \star E) = \mathbf{g}^{a+b} \star E$ , which we can compute from the knowledge of  $a$  and  $\mathbf{g}^b \star E$ , even without knowing  $b$ . On the other hand, it is hard to compute  $\mathbf{g}^{a+b} \star E$  from the knowledge of  $\mathbf{g}^a \star E$  and  $\mathbf{g}^b \star E$  only [22]. *Multiplication* in the exponent of the group generator, on the other hand, does not straightforwardly follow in the same way. In fact, there is no direct way to compute  $\mathbf{g}^{ab} \star E$  from knowledge of  $a$  and  $\mathbf{g}^b \star E$ , let alone from the knowledge of  $\mathbf{g}^a \star E$  and  $\mathbf{g}^b \star E$ ; this extends to exponentiation.

The possibility of using more complex arithmetic operations in the exponent of the generator is what makes the discrete logarithm setting so versatile. The goal of this work is to extend the flexibility of CSIDH-based schemes to also incorporate these operations and hopefully allow to construct protocols that appear out of reach with the current toolbox. We note that these new operations come with new hard problems. These problems also can be used to construct new zero-knowledge protocols based on them, which are explored in this work.

**Our contributions.** We extend the current isogeny-based group action toolbox with the possibility of performing squaring and (scalar) multiplications in the exponent of a public class group generator  $\mathbf{g}$  acting on elliptic curves. If this generator is public, we can define the group action using the group in the exponent of the group generator. Assuming  $\langle \mathbf{g} \rangle = \mathcal{G}$  of order  $N$ , this group action looks like

$$\begin{aligned} [\ ] : \mathbb{Z}/N\mathbb{Z} \times \mathcal{E} &\rightarrow \mathcal{E} \\ (a, E) &\mapsto [a]E = \mathbf{g}^a \star E. \end{aligned}$$

Table 1 presents a brief overview of the different arithmetic operations we define on isogenies. In order to draw some parallels to the discrete log (DLOG) setting, we define  $\mathcal{H}$  to be a group in which the discrete logarithm problem is hard. We can describe operations in the DLOG setting as a group  $\mathcal{D}$  acting on  $\mathcal{H}$  with  $\star : \mathcal{D} \times \mathcal{H} \rightarrow \mathcal{H}$  as  $\mathbf{a} \star h \mapsto h^{\mathbf{a}}$ . The DLOG assumption implies that it is hard to find  $\mathbf{a} \in \mathcal{D}$ , given  $(h, h^{\mathbf{a}})$ . If  $\mathcal{D}$  is generated by a fixed  $\mathbf{g} \in \mathcal{D}$  and  $\#\mathcal{D} = M$ , we can also define the group action  $[\ ] : \mathbb{Z}_M \times \mathcal{H} \rightarrow \mathcal{H}$ , such that  $[a]h \mapsto \mathbf{g}^a \star h = h^{\mathbf{g}^a}$ . Note that we do not assume the discrete logarithm problem to be hard in  $\mathcal{D}$ .

After introducing some background in Section 2, we start Section 3 by introducing new hardness assumptions related to these new operations. We show how they can be reduced to known problems in the literature. We then discuss two-party protocols for multiplications of the type  $[ab]E = \mathbf{g}^{ab} \star E$ , where one

**Table 1.** Arithmetic operations for the discrete logarithm and isogeny settings. We show how to compute operations given the minimal amount of secret information, and indicate the minimum number of secrets necessary to do so. For better illustration, we abuse notation by using the same letters for elements in  $\mathcal{G}$  and  $\mathcal{D}$ , as well as in  $\mathbb{Z}_N$  and  $\mathbb{Z}_M$ . In both cases, we assume  $\mathbf{a}$  and  $\mathbf{b}$  to be secret and  $c$  and  $e$  to be public scalars. We identify  $\mathbf{a} = \mathbf{g}^a$  and  $\mathbf{b} = \mathbf{g}^b$ , and assume  $\mathbf{a} \star E$ ,  $\mathbf{b} \star E$ ,  $h^a$ ,  $h^b$  to be public.

	Isogenies	Discrete logarithm	Secrets
Addition	$\mathbf{a} \star (\mathbf{b} \star E) = \mathbf{b} \star (\mathbf{a} \star E) = \mathbf{ab} \star E$ $= [a][b]E = [b][a]E = [a + b]E$	$(h^a)^b = (h^b)^a = h^{ab} =$ $(h^{g^a})^{g^b} = (h^{g^b})^{g^a} = h^{g^{a+b}}$	1
Scalar Mult.	$\mathbf{a}^c \star E = [ca]E$	$h^{a^c} = h^{g^{ca}}$	1
Exponentiation	$\mathbf{g}^{a^e} \star E = [a^e]E$	$h^{g^{a^e}}$	1
Multiplication	$\mathbf{a}^b \star E = \mathbf{b}^a \star E = [ab]E$	$h^{a^b} = h^{b^a} = h^{g^{ab}}$	2

party knows  $a$ , and the other  $b$ , without revealing those elements. We note that interactive protocols of this type need a trusted setup, which we also introduce.

In Section 4, we then present zero-knowledge proof systems for the languages defined by our newly proposed hardness assumptions. These protocols allow for example to prove that some tuples, such as  $(E, [a]E, [b]E, [ab]E)$  or  $(e, E, [a]E, [a^e]E)$  are well-formed. We achieve this in two ways, either with standard sigma protocols, or using MPC-in-the-Head (MPCitH). These proofs are competitive with protocols proving correct structure of additive tuples, i.e. tuples of the form  $(E, [a]E, [b]E, [a + b]E)$  as they were presented by Cozzo and Smart [26]. Again, our protocols rely on a trusted setup. In the MPCitH protocols, however, we can remove the trusted setup by using the “cut-and-choose” technique [47]. We show that proofs of multiplication and exponentiation are reminiscent of pairings in elliptic curve cryptography, although without the advantage of public verifiability.

We conclude this work by introducing a new post-quantum secure oblivious pseudo-random function (OPRF) in Section 5, based on the new tools presented in this work. Our OPRF relies on a trusted setup which can be removed using a pre-processing protocol based on OT extensions [6, 48, 49], and relies on a new hardness assumption, which we motivate well. Compared to the state-of-the-art of post-quantum OPRFs, our protocol has very competitive computational and communication cost, even for conservative security parameters, and is round-optimal. Furthermore, we can extend our construction to a verifiable OPRF.

**Related work.** Boneh, Kogan and Woo [17] introduced the first two OPRF constructions based on isogenies, one based on SIDH and one on CSIDH assumptions. Although the SIDH-based OPRF was first broken by Basso et al. [9] and later SIDH itself [21, 52, 59], a new design by Basso [8] solves both of these problems, at the cost of working with much larger parameters. Independently, Heimberger et al. modified the CSIDH-based OPRF of Boneh et al. to also work in the restricted effective group action setting [42] (where canonical representation of class group elements is not possible [2]), while also decreasing the

computational and communication cost of the protocol. On the downside, this protocol has a much higher round complexity.

Independently of this work, Joux [45] proposed MPC-in-the-head protocols to decrease the soundness error of the CSI-FiSh identification protocol [16]. While we are also considering MPC-in-the-head protocols, we note that there is no direct overlap between this work and theirs, as we are introducing new types of identification schemes.

## 2 Background

*Notation.* We let  $\lambda$  denote the computational security parameter. A function  $f(x)$  is *negligible* if for any constant  $c$ , there exists  $x_0$  such that for all  $x > x_0$ , we have  $f(x) < x^{-c}$ . We write  $\mathbb{Z}_N = \mathbb{Z}/N\mathbb{Z}$  and  $[n] = \{1, \dots, n\}$ .

### 2.1 Isogeny-based cryptography

Isogenies are surjective morphisms between elliptic curves. Endomorphisms are isogenies from elliptic curves to themselves. The endomorphisms of an elliptic curve  $E$ , together with the zero-map, define a ring under addition and composition. For supersingular elliptic curves over  $\mathbb{F}_p$ , the ring of  $\mathbb{F}_p$ -rational endomorphisms is isomorphic to an order  $\mathcal{O}$  in the quadratic imaginary field  $\mathbb{Q}(\sqrt{-p})$ .

Separable isogenies are uniquely determined by their kernel, which can be expressed as a finite group of points on the domain. A straightforward way to generate such a group of points is via the kernel of ideals  $\mathfrak{a} \in \mathcal{O}$ , then  $E[\mathfrak{a}] = \{P \in E(\overline{\mathbb{F}_p}) \mid \forall \alpha \in \mathfrak{a} : \alpha(P) = \infty\}$  defines the kernel of an isogeny  $E \rightarrow \mathfrak{a} \star E := E/E[\mathfrak{a}]$ . We can interpret elements in the ideal-class group  $\text{cl}(\mathcal{O})$  as acting on the set of supersingular elliptic curves over  $\mathbb{F}_p$  whose  $\mathbb{F}_p$ -rational endomorphism ring is isomorphic to  $\mathcal{O}$ . We denote this set as  $\mathcal{E}$  and write the action as  $\star : \text{cl}(\mathcal{O}) \times \mathcal{E} \rightarrow \mathcal{E}$ . We note that this action is free and transitive. As a shorthand, we will simply refer to ideals in the class groups as isogenies. If the class group is cyclic and a generator  $\mathfrak{g}$  is known and fixed, we can write the group action using the exponent notation introduced in [16],

$$\begin{aligned} [\ ] : \mathbb{Z}_N \times \mathcal{E} &\rightarrow \mathcal{E} \\ (a, E) &\mapsto [a]E = \mathfrak{g}^a \star E, \end{aligned}$$

where  $N = \#\text{cl}(\mathcal{O})$ .<sup>1</sup> Note that  $\mathbb{Z}_N$  defines the group in the exponent of the generator  $\mathfrak{g}$ , e.g. consecutive actions amount to the addition of the elements in  $\mathbb{Z}_N$ , e.g.  $[a]([b]E) = \mathfrak{g}^a \star (\mathfrak{g}^b \star E) = \mathfrak{g}^{a+b} \star E = [a+b]E$ .

It is important to note that the class number  $N$  is generally a composite number. This implies that the coefficients in the exponent are defined over a ring, rather than a field as is commonly the case in cryptographic constructions.

<sup>1</sup> We assume  $\text{cl}(\mathcal{O})$  to be cyclic. While not always the case, we can work in a cyclic subgroup of a non-cyclic class group. With high probability, such large cyclic subgroups exist [24]. For the rest of this work, we assume that  $N$  is not a smooth number.

While this does not really impact addition, we have to be more careful when talking about multiplication, especially concerning elements that do not have a multiplicative inverse.

Throughout this work, we consider supersingular elliptic curves defined over some prime field  $\mathbb{F}_p$ . For efficiency reasons (see [22]),  $p$  is chosen to have the form  $p = 4 \prod_{i=1}^n \ell_i - 1$ , where  $\ell_1, \dots, \ell_{n-1}$  are the  $n - 1$  smallest distinct odd primes and  $\ell_n > \ell_{n-1}$  is the smallest possible prime that makes  $p$  a prime as well. By this choice, the ideal  $\ell_i \mathcal{O}$  always splits into a prime ideal  $\mathfrak{l}_i$  and its conjugate  $\overline{\mathfrak{l}}_i$ , defining an isogeny of degree  $\ell_i$  or its dual. Now, the consecutive computation of many small isogenies of these prime degrees allows to compute isogenies of large degree efficiently, even if the class group structure is unknown. However, translating any arbitrary isogeny, e.g.  $\mathfrak{g}^a$  into small prime degree isogenies, requires knowing the relation lattice between the different elements  $\mathfrak{l}_1, \dots, \mathfrak{l}_n$ . Then arbitrary elements can be reduced modulo this lattice to yield efficiently computable isogenies, although the actual efficiency depends on how short the basis of the relation lattice is. The class group structure and a short relation lattice have been computed for the CSIDH-512 parameter set [16] (where  $\log p \approx 512$ ). Higher parameter sets can be computed in polynomial time on a quantum computer [50] or with the approach outlined by De Feo et al. [27], but the reduction of the relation lattices are less efficient. A workaround for higher parameter sets can be found in [56]. Throughout this work, we assume that uniform sampling and canonical representations of elements as powers of a generator are possible and efficient. Most notably, this allows to construct efficient signatures that do not need rejection sampling [28], such as CSI-FiSh [16].

In the ID-protocol underlying CSI-FiSh, a prover proves knowledge of a secret element  $s$  connecting two elliptic curves  $E_0$  and  $E_1 = [s]E_0$ , which is a witness for the following hard problem.

*Problem 1 (Group action inverse problem (GAIP)).* Given a pair of elliptic curves  $(E_0, E_1) \in \mathcal{E}^2$ , find  $s \in \mathbb{Z}_N$ , such that  $[s]E_0 = E_1$ .

To prove knowledge of  $s$ , the prover commits to a curve  $E_b = [b]E_0$  for a random  $b \leftarrow \mathbb{Z}_N$ , then upon receiving a challenge  $c \leftarrow \{0, 1\}$ , returns  $r = b - cs$ . Finally, the verifier accepts the proof only if  $[r]E_c = E_s$ .

## 2.2 Zero-Knowledge Proofs

Let  $X$  and  $W$  be sets, and let  $\mathcal{R} : X \times W \rightarrow \{0, 1\}$  be a relation defining the language  $\mathcal{L} = \{x \in X : \exists w \in W \text{ with } \mathcal{R}(x, w) = 1\}$ . First introduced by Goldwasser, Micali and Rackoff, interactive proofs are two-party protocols where a prover  $P$  convinces a verifier  $V$  that, given  $x \in X$ , it knows a witness  $w \in W$  such that  $\mathcal{R}(x, w) = 1$  [41]. Sigma protocols are protocols that execute in three rounds, in which the prover first sends a commitment value  $b$  to  $V$ , who then issues a challenge  $c$ . The prover responds to the challenge with  $r$ . After a verification step, the verifier then either accepts or rejects the proof. In order to be secure, such an interactive proof must be *complete* and *sound*. Completeness implies that if  $\mathcal{R}(x, w) = 1$ , the honest verifier accepts the proof, while soundness

requires that a malicious prover cannot make an honest verifier accept a proof for a relation  $\mathcal{R}(x', w') = 0$ , up to negligible probability. The stronger notion of *2-special soundness* requires the existence of a PPT algorithm (the extractor), which, given two accepting transcripts for the same commitment  $b$ , but with different challenges  $c \neq c'$ , is able to extract a witness  $w$ . Optionally, interactive proofs can also be (*honest verifier*) *zero-knowledge*, which implies that a (honest) verifier cannot extract any information about the witness  $w$  from the transcript of the protocol. This last property can be proven by building a simulator that can produce a protocol transcript that is indistinguishable from a real one, without the knowledge of  $w$ .

**Definition 1 (Completeness).** *A sigma protocol between parties  $P, V$  is complete for the relation  $\mathcal{R}$ , if for all  $\mathcal{R}(x, w) = 1$ ,  $V$  outputs True.*

**Definition 2 (Special soundness).** *A sigma protocol between parties  $P, V$  is special sound, if there exists a PPT extractor  $\text{Ext}$ , which for any  $x \in \mathcal{L}$ , when given two valid protocol transcripts  $(b, c, r)$  and  $(b, c', r')$ , can extract  $w \in W$ , such that  $\mathcal{R}(x, w) = 1$ .*

**Definition 3 (Zero-knowledge).** *A sigma protocol between parties  $P, V$  is zero-knowledge, if there exists a PPT simulator  $\text{Sim}$ , which for any  $x \in \mathcal{L}$  can generate a transcript  $(b, c, r)$  of the protocol, indistinguishable from a real transcript of the protocol, without knowledge of  $w$ .*

**MPC-in-the-Head.** First proposed by Ishai, Kushilevitz, Ostrovsky and Sahai in 2007, the MPC-in-the-Head (MPCitH) framework is a recent construction for zero-knowledge proof systems using secure multiparty computation (MPC) [44].

To prove a relation  $\mathcal{R}(x, w) = 1$ , the prover simulates “in its head” the execution of an  $n$ -party protocol which verifies the witness. To this end, the prover samples random tapes for each party, gives them their share  $w_i$  of the witness, and simulates their communication and internal states according to the MPC protocol. With the random tape  $r_i$ , the witness share  $w_i$ , and all the incoming messages, the prover can save the internal view of party  $P_i$ . After receiving commitments to each of these views, the verifier selects a subset at random which the prover has to open. The verifier then checks for each opened views that the incoming and outgoing messages are consistent with each other, and that the local computations have been performed correctly and finally, if the execution of the MPC protocol yields  $\mathcal{R}(x, w) = 1$ . Based on this, the verifier accepts or rejects the proof.

This work focuses on MPCitH proofs built from MPC protocols that use additive secret-sharing and that are secure against passive corruptions. Assuming ideal commitments, this implies that the resulting proof system is zero-knowledge when the verifier opens  $n - 1$  of the committed views, and it has soundness error  $1/n$ , since the prover can successfully guess the verifier’s challenge and cheat on one out of the  $n$  views. More formally, the MPC protocol  $\Pi$  should satisfy the following definition of privacy, with  $t = n - 1$ .

**Definition 4 (*t*-privacy [44]).** Let  $1 \leq t < n$ . We say that  $\Pi$  realizes  $f$  with *t*-privacy if there is a PPT simulator  $\text{Sim}$  such that for any inputs  $x, w_1, \dots, w_n$  and every set of corrupted players  $T \subseteq \{1, \dots, n\}$ , where  $|T| \leq t$ , the joint view  $\text{View}_T(x, w_1, \dots, w_n)$  of players in  $T$  is indistinguishable from  $\text{Sim}(T, x, (w_i)_{i \in T}, f_T(x, w_1, \dots, w_n))$ .

We refer the reader to previous works on MPCitH for the formal security statements of generic proofs built within this paradigm and for the reduction of the soundness error using repetition [44, 47].

**The cut-and-choose technique.** The cut-and-choose method is used to provide security against malicious adversaries. When a single party has to generate commitments to sets of correlated randomness, but is not trusted to do so honestly, they are asked to provide commitments to additional sets of randomness. Then, the other member(s) of the computation will randomly “cut and choose” some of the commitments that the generator must open to demonstrate their honesty. Since the generating party does not know in advance which commitments will be “cut”, and which will be “chosen” for the computation, this serves as a probabilistic test for the correctness of the unopened sets of randomness.

In the case of MPCitH argument systems, using the cut-and-choose technique was first proposed by Katz, Kolesnikov and Wang to enable the use of MPC protocols with a witness-independent pre-processing phase [47]. Concretely, the idea is that, before simulating the views of the parties during the MPC protocol, the prover commits to  $m$  executions of the pre-processing generation, of which the verifier cuts  $m - \tau$  in order to verify that they were correctly computed. If the prover has cheated on  $c \in [m]$  of these executions, then the probability that the cut-and-choose technique fails to uncover this is  $\binom{m-c}{m-\tau} / \binom{m}{m-\tau}$ .

For each of the  $\tau$  executions chosen after the cut, the prover has probability  $1/n$  of successfully cheating if the pre-processed data is honest, and has probability 1 otherwise. Accounting for the  $c$  executions with dishonest pre-processing, this implies that the Prover has probability  $1/n^{\tau-c}$  of successfully cheating in the online phase of the MPC protocol.

In summary, MPCitH proof systems based on the cut-and-choose technique have the following soundness error formula [11, Theorem 2]:

$$\epsilon_{\text{CnC}}(m, n, \tau) = \max_{0 \leq c \leq \tau} \left\{ \frac{\binom{m-c}{m-\tau}}{\binom{m}{m-\tau} \cdot n^{\tau-c}} \right\}.$$

Furthermore, they require that the prover pre-computes  $m$  copies of the correlated randomness and that the verifier re-computes  $m - \tau$  of them.

**Non-interactive MPCitH proofs.** Generic MPCitH proof systems are always public-coin and secure in the honest-verifier setting. Therefore, the proof systems constructed in the original 3-round framework of Ishai et al. [44] can be transformed into non-interactive zero-knowledge proofs of knowledge, and therefore digital signature schemes, using the generic Fiat–Shamir transform [35]. The

MPCitH proof systems constructed in the cut-and-choose framework introduced by Katz et al. are usually 5-round protocols (or sometimes more) and can also be made non-interactive using recent techniques [10, 46, 47].

### 3 Towards multiplication from addition

This section introduces our toolbox to construct the protocols that follow in later sections. We start with the functionality to generate trusted tuples and show how this can be used by two parties to multiply their secrets.

#### 3.1 Tuple generation functionality

In 1991, Beaver introduced a method to securely compute the product of two secret values using a pre-computed triple of *random* secret values  $B = (x, y, z)$  such that  $z = xy$  [12]. When the elements of Beaver triples are additively secret-shared between multiple parties, it allows them to compute the product of two other secret-shared values, say  $a$  and  $b$ , by first opening masked values and then constructing the product  $c = ab$  using only local linear operations.

More precisely, let  $(a_i, b_i, x_i, y_i, z_i)$  be the additive shares that party  $P_i$  has of  $(a, b, x, y, xy)$ , respectively. By revealing their shares  $\alpha_i = a_i - x_i$  and  $\beta_i = b_i - y_i$ , the parties can each compute the values  $\alpha = \sum_i \alpha_i = a - x$  and  $\beta = \sum_i \beta_i = b - y$ .<sup>2</sup> Since  $x$  and  $y$  are random, this does not reveal information about  $a$  and  $b$ . Then, the parties can locally compute their additive share of  $c = a \cdot b$  by setting

$$c_i = \alpha y_i + \beta x_i + z_i + \begin{cases} \alpha\beta & \text{if } i = n, \\ 0 & \text{otherwise,} \end{cases}$$

It now follows that  $\sum_{i=1}^n c_i = \alpha y + \beta x + xy + \alpha\beta = (\alpha + x)(\beta + y) = ab$ . We can explicitly write the secret-shared triple as

$$B = (\text{id}; (x_1, \dots, x_n), (y_1, \dots, y_n), (z_1, \dots, z_n)) \in \{0, 1\}^* \times (\mathbb{Z}_N^n)^3, \quad (1)$$

with  $\sum_{i=1}^n z_i = (\sum_{i=1}^n x_i)(\sum_{i=1}^n y_i)$ , and where  $\text{id} \in \{0, 1\}^*$  is a unique identifier. We call *view* the set of shares or elements that a party has knowledge of. In general multi-party protocols as described above, the view of party  $P_i$  is given by  $\text{View}_{P_i}(B) = (\text{id}; x_i, y_i, z_i)$ .

*Multiplying group elements.* We show how to use Beaver triples of the type (1), when we want to compute the product of two secret-shared values  $a$  and  $b$  in  $\mathbb{Z}_N$ . Again, let  $\alpha = a - x$  and  $\beta = b - y$  and assume they are given to each of the  $n$  parties, as if they had been opened. Now,

$$[ab]E = \left[ \left( \alpha + \sum_{i=1}^n x_i \right) \left( \beta + \sum_{i=1}^n y_i \right) \right] E$$

<sup>2</sup> We note that all operations are operations in  $\mathbb{Z}_N$ , i.e. should be read modulo  $N$ .



$$\begin{aligned}
&= \left[ \alpha\beta + \alpha \sum_{i=1}^n y_i + \beta \sum_{i=1}^n x_i + \sum_{i=1}^n z_i \right] E \\
&= [\alpha y_1 + \beta x_1 + z_1] \cdots [\alpha y_n + \beta x_n + z_n + \alpha\beta] E,
\end{aligned}$$

and parties can compute  $[ab]E$  by each  $P_i$  computing the action  $[\alpha y_i + \beta x_i + z_i]$  in a round-robin fashion, using only their knowledge of  $\alpha, \beta$  and  $View_{P_i}(B)$ . The final action  $[\alpha\beta]E$  is computed by  $P_n$  (but could in fact be computed by any party). We summarize the round-robin steps in Figure 1.

---

MPC multiplication protocol for  $P_i(\alpha, \beta, x_i, y_i, z_i)$

```

1 : from  $P_{i-1}$  receive  $F_{i-1}$  ( $P_1$  receives  $F_0 = E_0$ )
2 : if  $i < n$  then
3 :    $F_i \leftarrow [\beta x_i + \alpha y_i + z_i] F_{i-1}$ 
4 :   send  $F_i$  to  $P_{i+1}$ 
5 : if  $i = n$  then
6 :    $F_n \leftarrow [\alpha\beta + \beta x_n + \alpha y_n + z_n] F_{n-1}$ 
7 :   Broadcast( $F_n$ )

```

**Fig. 1.** Round-robin step in the multiplication protocol for  $n$  parties.

This protocol is only passively secure, i.e. a correct output is only guaranteed if all parties behave according to the instructions of the protocol. In order to make this protocol actively secure, we introduce zero-knowledge proofs later in Section 4, which parties can append to their messages to prove that their computations have been indeed performed correctly. But this protocol is even more versatile: we further show in Section 4, how it can be used to create MPCitH proof systems in order to prove correct multiplication by a party that knows  $a$  and  $b$ .

Note that the way that  $\alpha$  and  $\beta$  are communicated to the parties varies with each of those protocols. Sometimes, they will be generated and shared by the parties themselves as in the protocol above, sometimes by an external dealer (e.g. for MPCitH protocols where the Prover can deal  $\alpha$  and  $\beta$  to the parties as global inputs, see Sections 2.2 and 4.4)

*Tuple generation functionality.* Equation (1) is just an example of a Beaver triple sharing, in this case for  $n$ -party multiplication with passive security. Throughout this work, we will need more general *tuples*, which will look different depending on the arithmetic operation to be performed or the assumptions about the adversaries. In particular, we will sometimes need extra elements, such as specific elliptic curves, as part of the tuples. These different shapes and structures will be introduced in the relevant sections. We now define the generalized functionality used in this work to represent these tuples, and parties' access to them. What *valid* means, will depend on the context and be introduced whenever needed.

**Definition 5 (Tuple generation functionality).** Let  $\mathcal{T}$  be an algorithm that generates valid tuples of a predetermined type for a set of parties  $\mathcal{P} = \{P_1, \dots, P_n\}$ . On input  $\text{id}$  by party  $P$ , if  $P \notin \mathcal{P}$ ,  $\mathcal{T}$  returns  $\perp$ . If no such tuple exists,  $\mathcal{T}$  generates a new tuple  $B$  with identifier  $\text{id}$ . In either case, if  $P \in \mathcal{P}$ ,  $\mathcal{T}$  returns  $\text{View}_P(B)$ .

In the simplest case, we will assume that  $\mathcal{T}$  is a trusted party, and that the parties engaging in protocols needing such tuples have black-box access to  $\mathcal{T}$ . In some cases (e.g. for MPCitH) however, we can use the *cut-and-choose* technique to let the prover generate them (see Section 2.2). We note that these tuples are independent of the secret pair to be masked and as such, can be precomputed.

### 3.2 Two-party multiplication protocol

Suppose we have two parties  $P_a$  and  $P_b$ , which using their respective secrets  $a, b \in \mathbb{Z}_N$  want to compute  $[ab]E_0$ , where  $E_0$  is a specific starting curve. We describe an interactive 2-party protocol between  $P_a$  and  $P_b$  to compute  $[ab]E_0$  together, using precomputed tuples by a trusted third party  $\mathcal{T}$ . The generated tuples have the following form

$$B = (\text{id}; x, y, z_a, z_b) \in \{0, 1\}^* \times \mathbb{Z}_N^4, \\ \text{View}_{P_a}(B) = (\text{id}, x, z_a), \quad \text{and} \quad \text{View}_{P_b}(B) = (\text{id}, y, z_b).$$

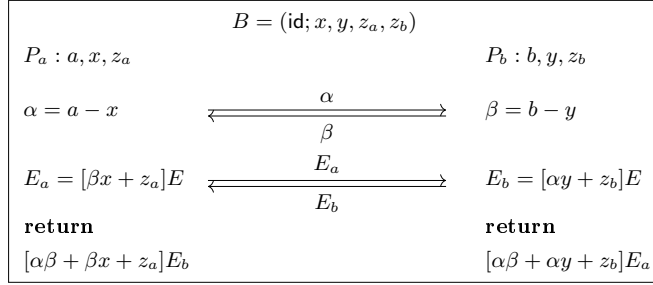
with  $z_a + z_b = xy$ . Inspired by traditional MPC protocols, the parties first secret-share their inputs  $a$  and  $b$  with each other as  $\alpha = a - x$  and  $\beta = b - y$  and then combine these sharings with secret information to compute partial curves, which can be used to each reconstruct the desired curve  $[ab]E_0$ . As such, we can also see this protocol as an interactive Diffie–Hellman key exchange. Figure 2 details this joint multiplication protocol for two parties, which uses a tuple  $B$  of the aforementioned form as input. The evaluation of this protocol costs 2 group actions per player (performed in parallel), which is only twice the cost of the Diffie–Hellman protocol of Castryck et al. [22].<sup>3</sup>

**Theorem 1.** *The protocol of Figure 2 is correct and private.*

*Proof. Correctness:* This follows from the structure of the tuple.

**Privacy:** Let  $\mathcal{S}_b$  be a simulator interacting with the party  $P_a$ . Whenever  $P_a$  requests a new tuple,  $\mathcal{S}_b$  generates  $B = (\text{id}; x, y, z_a, z_b)$  and sends  $\text{View}_{P_a}(B)$  to  $P_a$ . Upon reception of the output curve  $E_{\text{out}}$ ,  $\mathcal{S}_b$  first samples  $\beta \leftarrow \mathbb{Z}_N$  and sends it to  $P_a$  in the first round. In the second round,  $\mathcal{S}_b$  computes  $E_b = [-(\alpha\beta + \beta x + z_a)]E_{\text{out}}$  and sends it to  $P_a$ . The transcript generated by  $\mathcal{S}_b$  is indistinguishable from real transcripts, as both the real and simulated  $\beta$  are uniformly random in  $\mathbb{Z}_N$  and both  $E_b = [r]E_{\text{out}}$ , for some  $r$  uniformly random in  $\mathbb{Z}_N$ . The simulator  $\mathcal{S}_a$  interacting with  $P_b$  works exactly the same.  $\square$

<sup>3</sup> We can remove the symmetry of this protocol and make e.g. party  $P_a$  only compute  $E_a$ , then send it to  $P_b$ , which computes the output  $[ab]E$ . Then, each party only has to perform a single group action, while also reducing the communication cost between the parties. Still, the total cost of the protocol remains 2 group actions.



**Fig. 2.** Passively secure 2-party multiplication protocol from additive secret sharing, using a trusted tuple  $B$  as input.

## 4 Zero-Knowledge Proof Systems

In this section, we present zero-knowledge proof systems for different arithmetic operations in  $\mathbb{Z}_N$ . Among other applications, these will allow us to augment the passively secure protocol from the previous section to active security. We distinguish proofs along two dimensions: the relation that they prove (addition, scalar multiplication, exponentiation or secret multiplication) and the paradigm that they follow (from computational assumptions or from MPC-in-the-Head).

### 4.1 Languages and Security Assumptions

We consider the following languages, with  $E, E', E_i, E'_i \in \mathcal{E}$ ,  $a, b, c_i \in \mathbb{Z}_N$  and  $e_i \in \mathbb{N}$ :

$$\mathcal{L}_k^{\text{Add}} = \left\{ ((E_i, E'_i)_{i=1, \dots, k}, a) : \bigwedge_{i=1}^k E'_i = [a]E_i \right\} \quad (2)$$

$$\mathcal{L}_k^{\text{Scal}} = \left\{ ((E_i, E'_i, c_i)_{i=1, \dots, k}, a) : \bigwedge_{i=1}^k E'_i = [c_i a]E_i \right\} \quad (3)$$

$$\mathcal{L}^{\text{Exp}} = \left\{ ([a]E, E', e, a) : E' = [a^e]E \right\} \quad (4)$$

$$\mathcal{L}^{\text{Mult}} = \left\{ ([a]E, [b]E, E'), (a, b) : E' = [ab]E \right\} \quad (5)$$

A zero-knowledge protocol for  $\mathcal{L}_k^{\text{Add}}$  already exists [26], while a version of  $\mathcal{L}_k^{\text{Scal}}$  was proposed by Atapoor et al. [4], but with  $c_1 = 1$  fixed. Both languages coincide for  $c_1 = \dots = c_k = 1$ , thus  $\mathcal{L}_k^{\text{Add}}$  can be seen as a special case of  $\mathcal{L}_k^{\text{Scal}}$ . We can see that  $\mathcal{L}_1^{\text{Add}}$  is exactly the language underlying the ID protocol of CSI-FiSh outlined in Section 2.1, with GAIP (Problem 1) the associated security assumption. For  $\mathcal{L}_2^{\text{Add}}$ , the underlying assumption corresponds to the computational Diffie–Hellman problem.

*Problem 2 (Computational Diffie-Hellman problem (CDH) [22, 25, 60]).* Given  $(E, [a]E, F)$ , where  $a \in \mathbb{Z}_N$  and  $E, F \in \mathcal{E}$ , compute  $[a]F$ .

We can define similar computational problems when looking at scalar multiplication, i.e. for  $\mathcal{L}_1^{\text{Scal}}$ , we can define

*Problem 3 (Scalar-CDH [7, 34]).* Given  $(c, E, [a]E)$  where  $c, a \in \mathbb{Z}_N$  and  $E \in \mathcal{E}$ , compute  $[ca]E$ .

For our purposes, these assumptions cover the hardness of executing additions or scalar multiplications with unknown secrets. Increasing  $k$  would define more hardness assumptions but will not be relevant in this work. Instead, we introduce the following new assumptions, which also cover exponentiation and multiplication, i.e. are related to  $\mathcal{L}^{\text{Mult}}$  and  $\mathcal{L}^{\text{Exp}}$ .<sup>4</sup>

*Problem 4 (Exp-CDH).* Given  $(E, [a]E, e)$ , where  $e \in \mathbb{N}$ ,  $a \in \mathbb{Z}_N$  and  $E \in \mathcal{E}$ , compute  $[a^e]E$ .

*Problem 5 (Mult-CDH).* Given  $(E, [a]E, [b]E)$ , where  $a, b \in \mathbb{Z}_N$  and  $E \in \mathcal{E}$ , compute  $[ab]E$ .

In Appendix A, we show that the following reductions hold when  $N$  is odd. This condition is guaranteed by choosing  $p \equiv 3 \pmod{4}$  [22].

$$\text{CDH} \equiv \text{Scalar-CDH} \leq \text{Exp-CDH} \equiv \text{Mult-CDH} \leq \text{GAIP}. \quad (6)$$

Note that following from [39, 53, 61], CDH and GAIP are equivalent under quantum reductions, implying that the hardness of all problems in this section collapse to the quantum hardness of GAIP.

## 4.2 Addition and Scalar Multiplication

In this section, we present the more general version of the protocol first introduced by Atapoor et al. [4]. This protocol is a proof system for the general scalar multiplication language  $\mathcal{L}_k^{\text{Scal}}$ , but a proof system for  $\mathcal{L}_k^{\text{Add}}$  can be obtained as discussed above. These protocols have furthermore been proven secure in the QROM [4, 13]. We note that the latter proof can straightforwardly be extended to the case  $c_1 \neq 1$ . We summarize the protocol in Figure 3. We note that the soundness error is  $2^{-\lambda}$ . Throughout this section, we define the cryptographic hash function  $H : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$ , where  $\lambda$  is a given security parameter.

*Cost.* The computational cost of the protocol is  $k\lambda$  group actions, for both the proof and also for the verification part, neglecting other costs, such as  $\mathbb{Z}_N$ -arithmetic and hash computations.

<sup>4</sup> We emphasize that the notation in [34] deviates from ours, as e.g. squaring in [34] is related to computing  $[2a]E$  from  $(E, [a]E)$ , which we call doubling.

<p>Scal.Prove<math>_{\lambda}(a, X)</math></p> <p><b>Input:</b> Secret <math>a</math>, set <math>X = \{(E_i, E'_i, c_i)\}_{i=1, \dots, k}</math>, s.t. <math>E_i, E'_i \in \mathcal{E}</math> and <math>\{c_1, \dots, c_k\}</math> an exceptional set modulo <math>N</math>, security parameter <math>\lambda</math>.<sup>5</sup></p> <p><b>Output:</b> A proof <math>\pi</math> for the language <math>\mathcal{L}_k^{\text{Scal}}</math>.</p> <ol style="list-style-type: none"> <li>1. For <math>j = 1, \dots, \lambda</math>: <ol style="list-style-type: none"> <li>(a) <math>b_j \leftarrow \mathbb{Z}_N</math></li> <li>(b) For <math>i = 1, \dots, k</math>, compute <math>\hat{E}_{i,j} = [c_i b_j] E_i</math></li> </ol> </li> <li>2. <math>d_1 \dots d_{\lambda} = \text{H}(X, \{\hat{E}_{i,j}\}_{i=1, \dots, k}^{j=1, \dots, \lambda}) \in \{0, 1\}^{\lambda}</math></li> <li>3. For <math>j = 1, \dots, \lambda</math>, compute <math>r_j = b_j - d_j a</math>.</li> <li>4. Return <math>\pi = ((d_1, \dots, d_{\lambda}), (r_1, \dots, r_{\lambda}))</math>.</li> </ol> <p>Scal.Verify(<math>X, \pi</math>)</p> <p><b>Input:</b> Statement <math>X = \{(E_i, E'_i, c_i)\}_{i=1, \dots, k}</math>, proof <math>\pi = ((d_1, \dots, d_{\lambda}), (r_1, \dots, r_{\lambda}))</math>.</p> <p><b>Output:</b> accept or reject</p> <ol style="list-style-type: none"> <li>1. For <math>j = 1, \dots, \lambda</math>: <ol style="list-style-type: none"> <li>– If <math>d_j = 0</math>, then let <math>\hat{E}_{i,j} = [c_i r_j] E_i</math> for <math>i = 1, \dots, k</math>,</li> <li>– If <math>d_j = 1</math>, then let <math>\hat{E}_{i,j} = [c_i r_j] E'_i</math> for <math>i = 1, \dots, k</math>.</li> </ol> </li> <li>2. Return <math>(d_1, \dots, d_{\lambda}) \stackrel{?}{=} \text{H}(X, \{\hat{E}_{i,j}\}_{i=1, \dots, k}^{j=1, \dots, \lambda})</math>.</li> </ol>
---

**Fig. 3.** Non-interactive zero-knowledge proof and verification for the language  $\mathcal{L}_k^{\text{Scal}}$ .

*Example.* As an application example, with  $c_1 = c_2 = 1$ , this protocol can be used to prove that four elliptic curves constitute a well-formed “Diffie-Hellman tuple”, i.e.  $(E, [a]E, [b]E, [a+b]E) \in \mathcal{L}_2^{\text{Add}}$  without revealing the secrets  $a$  and  $b$ , an observation that was initially proposed by Cozzo and Smart [26]. In the case where  $[a]E$  and  $[b]E$  are public elements, a prover needs to know only one of the secrets. Assuming for example that the prover knows  $a$  and  $[b]E$ , it can prove correctness of the tuple via

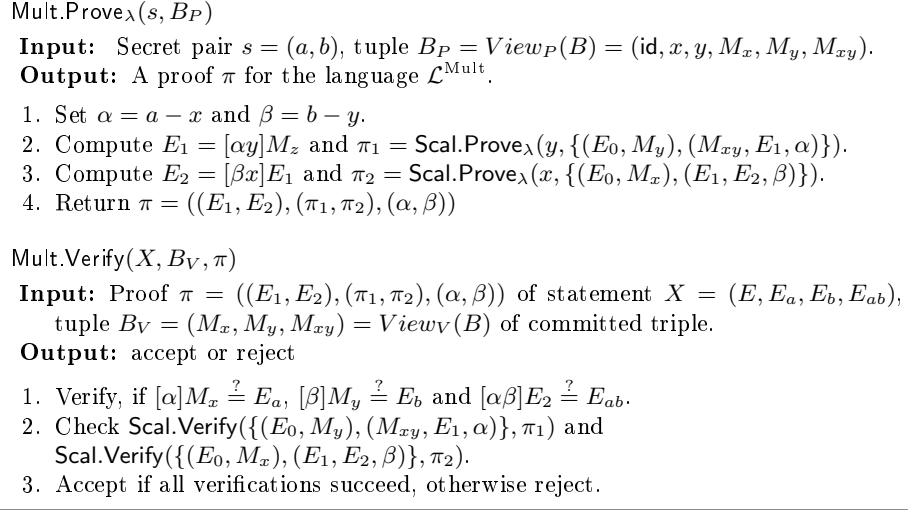
$$\pi \leftarrow \text{Scal.Prove}_{\lambda}(a, \{(E, [a]E), ([b]E, [a+b]E)\}),$$

where, for conciseness, we drop the  $c_i$  factor in the case where  $c_i = 1$ .

### 4.3 Multiplication with trusted setup

We use the scalar multiplication protocol from the previous section to create a protocol for general multiplication. This means that we assume a prover knowing  $a, b \in \mathbb{Z}_N$  wants to prove that a tuple  $(E, [a]E, [b]E, [ab]E) \in \mathcal{L}^{\text{Mult}}$ . The high level idea is that we can prove this in a way similar to the approach explained in Section 3.1. Initially, the prover discloses  $\alpha = a - x$  and  $\beta = b - y$  and computes the action of  $[ab]E = [(\alpha + x)(\beta + y)]E$  consecutively as  $[\alpha\beta][\beta x][\alpha y][xy]E$ . In order to prove that the individual actions have been computed correctly, the

<sup>5</sup> A set is exceptional modulo  $N$ , if the pairwise difference between any two elements is invertible in  $\mathbb{Z}_N$ . This ensures extractability for any challenge.



**Fig. 4.** Non-interactive zero-knowledge proof and verification for the language  $\mathcal{L}^{\text{Mult}}$  in the additive secret sharing case.

prover uses  $\text{Scal.Prove}$  at each step. As a reference for these proofs, the prover further needs commitments to the shares

$$M_x = [x]E_0, \quad M_y = [y]E_0, \quad M_{xy} = [xy]E_0.$$

Then, in order to prove correct execution of e.g. the action  $F' = [\alpha y]F$ , the prover runs  $\text{Scal.Prove}_\lambda(y, \{(E_0, M_y), (F, F', \alpha)\})$ . Using this idea, we show a protocol for the language  $\mathcal{L}^{\text{Mult}}$  in Figure 4. In order to guarantee that the commitments  $M_x, M_y, M_z$  have the correct structure, we assume that they are generated by the functionality  $\mathcal{T}$  of Definition 5. We require the tuples to have the form

$$B = (\text{id}; x, y, M_x, M_y, M_{xy}) \in \{0, 1\}^* \times \mathbb{Z}_N^2 \times \mathcal{E}^3, \quad (7)$$

with  $M_x, M_y, M_{xy}$  as above, such that the prover has full access to the elements, but only the  $M_i$  are public, i.e. accessible to the verifiers. We define their views as  $\text{View}_P(B) = B$  and  $\text{View}_V(B) = (\text{id}; M_x, M_y, M_{xy})$ .

**Theorem 2.** *The algorithms  $\text{Mult.Prove}$  and  $\text{Mult.Verify}$  realize a non-interactive zero-knowledge proof of knowledge for the language  $\mathcal{L}^{\text{Mult}}$ .*

*Proof. Completeness.* After correct execution of the protocol, we have

$$\begin{aligned} [\alpha]M_x &= [a - x][x]E_0 = [a]E_0, & [\beta]M_y &= [b - y][y]E_0 = [b]E_0, \\ & \text{and } [\alpha\beta]E_2 &= [\alpha\beta + \beta y + \alpha x + xy]E_0 = [ab]E_0. \end{aligned}$$

Furthermore, since  $M_y = [y]E_0$  and  $E_1 = [\alpha y]M_z$ , as well as  $M_x = [x]E_0$  and  $E_2 = [\beta x]E_0$ , both the verifications of  $\pi_1$  and  $\pi_2$  will succeed.

**Soundness.** Soundness of the full protocol is directly related to the soundness of `Scal.Proveλ` and `Scal.Verify`. We note that the extractor in either invocation of `Scal.Prove` allows the extraction of  $x$  or  $y$ , respectively. The secrets can then be recovered as  $a = \alpha + x$  or  $b = \beta + y$ . The total soundness error is  $2^{-\lambda}$ , given by the maximal soundness error of the different `Scal.Prove` subalgorithms.

**Zero-knowledge.** Zero-knowledge of our protocol immediately follows from the zero-knowledge property of `Scal.Prove` and the theorem of sequential composition for zero-knowledge [40, Theorem 9].  $\square$

*Remark 1.* Similarly to Figure 4, using `Scal.Prove` allows us to make protocols like Figure 2 actively secure. It suffices to attach a proof that the correct  $x$  or  $y$  has been used. To this end, the trusted tuples would also need to contain the curves  $M_x$  and  $M_y$ , which are accessible by both parties (but not  $M_{xy}$ ). These commitments also give parties the possibility to verify the received  $\alpha$  and  $\beta$ , by testing whether  $[\alpha]M_x \stackrel{?}{=} E_a$  and  $[\beta]M_y \stackrel{?}{=} E_b$ .

*Cost.* In Figure 4, the prover computes a total of  $2 + 4\lambda$  group actions and the verifier  $3 + 4\lambda$ . This is about twice the cost of proving correctness of DH-tuples in  $\mathcal{L}_2^{\text{Add}}$  using `Scal.Prove`, or verifying them using `Scal.Verify`.

**Exponentiation with trusted setup.** A similar idea to multiplication with Beaver triples can be used in order to prove elements in  $\mathcal{L}^{\text{Exp}}$ , e.g. compute powers of a secret  $a \in \mathbb{Z}_N$ , such as  $[a^e]E_0$ , for some  $e \in \mathbb{N}$ . In Appendix B.1, we introduce protocols for squaring and cubing, which can be combined to arbitrary exponents using a square-and-multiply approach.

#### 4.4 MPC-in-the-Head protocols

In this section, we propose alternative proofs of correct multiplication and exponentiation, using the MPCitH technique. These proofs again use tuples which have auxiliary elliptic curves in them, similar to the protocols in the previous section. If these tuples are generated by a trusted third party, such as the one in Definition 5, then the protocols in this section outperform the protocols presented previously, which also use such a trusted setup. But another advantage of the MPCitH technique is that we can use the cut-and-choose approach to remove the trusted third party. This results in slightly slower, but still competitive protocols that do not require a trusted setup.

**4.4.1 Multiplication-in-the-Head with trusted setup.** For the protocol in this section, we initially assume the existence of a trusted third party  $\mathcal{T}$ , accessible by both prover and verifier, to generate random sharings of tuples of the type

$$B = (\text{id}; (x_i, y_i, z_i)_{i \in [n]}, M_x, M_y) \in \{0, 1\}^* \times \mathbb{Z}_N^{3n} \times \mathcal{E}^2$$

for  $n$  parties, such that  $M_x = [x]E_0$  and  $M_y = [y]E_0$ , and where  $\sum_{i=1}^n x_i = x$ ,  $\sum_{i=1}^n y_i = y$  and  $\sum_{i=1}^n z_i = xy$ . Whenever the prover queries a new tuple, we let  $\mathcal{T}$  respond with  $B_P = (\text{id}; (x_i, y_i, z_i)_{i \in [n]})$ ; when the verifier queries  $\mathcal{T}$  for an existing tuple with identifier  $\text{id}$ , we let  $\mathcal{T}$  respond with  $B_V = (\text{id}; M_x, M_y)$  if  $\text{id}$  exists, and with  $\perp$  otherwise. As part of the proof, we assume that the prover can give the verifier access to tuples of the form  $B_i = (\text{id}; x_i, y_i, z_i)$ .

Once in possession of  $(x_i, y_i, z_i)_{i \in [n]}$ , the prover distributes these values among the  $n$  parties, together with the public values  $\alpha = a - x$  and  $\beta = b - y$ . In the same way as in Figure 1 (with the round-robin communication), the parties jointly compute  $F_n = [ab]E_0$ . We denote the joint execution of parties  $P_1, \dots, P_n$  as  $\text{MultiTH}_n(\alpha, \beta, B_P)$ .

To construct an interactive proof of knowledge from this MPC protocol, the prover commits to the views of the  $n$  parties (i.e. to their inputs and the messages they received) as well as the public values  $\alpha$  and  $\beta$ . We denote party  $i$ 's view as  $\mathcal{V}_i = (B_i, F_{i-1}, F_i)$ . In order to commit to these views, the prover samples secret values  $\mu_i \leftarrow \{0, 1\}^\lambda$  and computes the commitments  $C_i = \mathcal{C}(\mathcal{V}_i, \mu_i)$ . After receiving the commitments, the verifier responds with a random challenge  $c \in [n]$  which determines the party whose view the prover *does not* open. The prover therefore sends the set of  $n - 1$  views  $\{\mathcal{V}_i\}_{i \in [n] \setminus \{c\}}$  for which the verifier checks that (1)  $F_i$  has been correctly computed from  $F_{i-1}$  using  $\alpha$ ,  $\beta$  and  $B_i$ ; (2) the views are consistent with each other, i.e. if for all pairs  $\{i, i + 1\} \not\ni c$ , whether  $F_i$  contained in  $\mathcal{V}_i$  is consistent with  $F_i$  contained in  $\mathcal{V}_{i+1}$ ; (3a) in the case  $c \neq 1$ , the initial curve  $F_0$  is equal to  $E_0$  and (3b) in the case  $c \neq n$ , the final curve  $F_n$  implied by  $\mathcal{V}_n$  is equal to the expected outcome  $E_{ab}$  of the protocol.

As the challenge space from which  $c$  is sampled has size only  $n$ , this MPCitH protocol has soundness error  $1/n$  (see also Section 2.2) and must be repeated  $\tau = \lceil \lambda / \log_2 n \rceil$  times to achieve soundness error  $2^{-\lambda}$ . (See also Section 2.2 for the transformation into a non-interactive argument system.)

**Theorem 3.** *Assuming ideal commitments, the protocol in Figure 5 is an interactive and honest-verifier zero-knowledge proof of knowledge for the language  $\mathcal{L}^{\text{Mult}}$  with soundness error  $1/n$ .*

*Proof.* **Completeness:** The code in Figure 1 computes  $E_n = [ab]E_0$  via

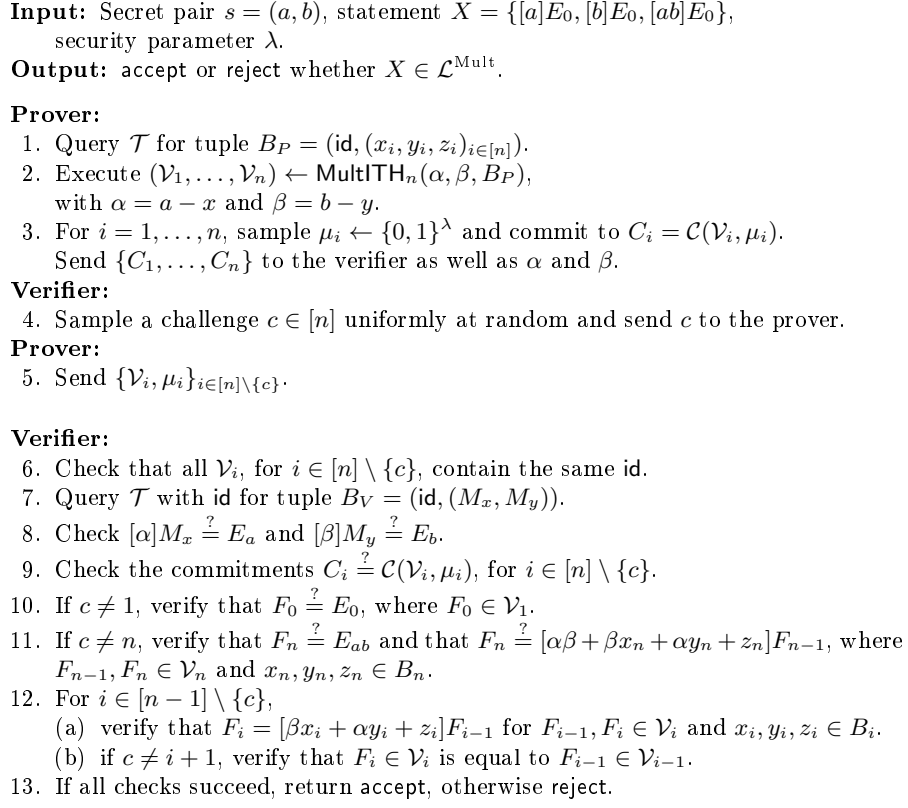
$$E_n = \left[ \alpha\beta + \beta \sum_{i=1}^n x_i + \alpha \sum_{i=1}^n y_i + \sum_{i=1}^n z_i \right] E_0 = [\alpha\beta + \beta x + \alpha y + xy] E_0.$$

We note that the checks in step 8 uniquely fix  $\alpha = a - x$  and  $\beta = b - y$ .

**Special soundness:** Let  $c$  and  $c'$  be two different challenges and let  $(\mathfrak{C}, c, \mathfrak{R})$  and  $(\mathfrak{C}', c', \mathfrak{R}')$  be two accepting transcripts, where  $\mathfrak{C} = (\{C_i\}_{i \in [n]}, \alpha, \beta)$  and  $\mathfrak{R} = (\{\mathcal{V}_i, \mu_i\}_{i \in [n] \setminus \{c\}})$ . By the binding property of the idealised commitment scheme, we have that  $B_i = B'_i$  for all  $i \in [n]$ , since  $C_c$  is verified as a valid commitment for  $\mathcal{V}'_c$  in  $\mathfrak{R}'$ . Thus, the extractor can reconstruct  $x$  and  $y$  from  $\{B_i\}_{i \in [n] \setminus \{c\}} \cup \{B'_c\}$  and recover  $a$  and  $b$  from  $\alpha$  and  $\beta$  respectively.

As discussed in Section 2.2, standard techniques then show that this protocol has soundness error  $1/n$  if executed once, and  $1/n^\tau$  if repeated  $\tau$  times.





**Fig. 5.** Interactive ZK proof for  $\mathcal{L}^{\text{Mult}}$  using MPC-in-the-Head with trusted party.

**Zero-knowledge:** The simulator  $\mathcal{S}$  plays the role of the prover (without knowledge of  $a$  or  $b$ ), of the trusted third party, and of the challenge generation in order to output a transcript that is indistinguishable from one made with a valid witness. (Note that the argument below is equivalent to proving the  $(n-1)$ -privacy of the protocol of Figure 1.)

First,  $\mathcal{S}$  samples  $\alpha$  and  $\beta$  at random. As the third party  $\mathcal{T}$ , it then creates a new tuple by sampling all  $x_i, y_i, z_i$  at random, for  $i \in [n]$  and setting  $M_x = [-\alpha]E_a, M_y = [-\beta]E_b$ , and creating a random id.

Next,  $\mathcal{S}$  samples  $c \in [n]$  at random on behalf of an honest verifier. Since  $\alpha$  and  $\beta$  contain no information about  $a$  and  $b$ , and the simulated triple is not correct, computing  $F_i$  honestly, for  $i \in [n]$ , will not produce  $F_n = E_{ab}$  as expected by the verifier. Instead, the simulator computes  $F_i$  for  $1 \leq i < c$  “forwards”, as per the protocol, but computes  $F_i$  for  $c \leq i \leq n$  “backwards” from  $F_n = E_{ab}$ ,

ensuring that the chain results in the correct final curve. Thus:

$$F_i = \begin{cases} [\beta x_i + \alpha y_i]F_{i-1} & 1 \leq i < c \\ [-\beta x_{i+1} - \alpha y_{i+1}]F_{i+1} & c \leq i < n-1 \\ [-\alpha\beta - \beta x_n - \alpha y_n]E_n & i = n-1, \end{cases}$$

where  $F_0 = E_0$ .

The simulator can now set  $\mathcal{V}_i = (\text{id}, x_i, y_i, F_{i-1}, F_i)$ , sample  $\mu_i$  as per the protocol, and compute the set of commitments  $\{C_i\}$ . This gives the transcript  $(\{C_i\}_{i \in [n]}, \alpha, \beta, c, \{\mathcal{V}_i, \mu_i\}_{i \in [n] \setminus \{c\}})$  as the final output of  $\mathcal{S}$ .

*Correctness:* The simulated transcript will verify since  $[\alpha]M_x = E_a$  and  $[\beta]M_y = E_b$  by construction;  $F_n = E_{ab}$  by construction of  $F_{n-1}$ ; all of the  $F_i$  curves, for  $i \neq c$ , satisfy the right relation.

*Indistinguishability:* Since  $x$  and  $y$  are uniformly generated by  $\mathcal{T}$  in the protocol, sampling  $\alpha$  and  $\beta$  directly in the simulation is perfectly indistinguishable. By the same argument, the curves  $M_x$  and  $M_y$  of the simulated tuple are distributed identically to an honest tuple, since in addition the idealised commitment perfectly hides  $x_c$  and  $y_c$  from the Verifier, who therefore cannot recover  $x$  and  $y$ . Within the MPC protocol, the only inconsistency is the computation of  $F_c$  which Verifier cannot detect since  $x_c$  and  $y_c$  are hidden by the commitment scheme.  $\square$

*Cost.* In Figure 1, each party computes one isogeny computation. The prover runs this protocol  $\tau$  times for  $n$  parties, resulting in a total computational cost of  $n\tau = n\lceil\lambda/\log_2 n\rceil$  for the prover. The verifier on the other hand has to verify the steps in Figure 1 for  $n-1$  parties and compute  $2\tau$  further isogeny computations in step 8 of the protocol, resulting in the total  $(n+1)\tau$ . By choosing  $n=3$ , we optimize with respect to the total cost of the proof and verification. In that case, the prover computes approximately  $1.89\lambda$  isogenies, and the verifier  $2.52\lambda$ .<sup>6</sup> We note that this cost is competitive with proofs of DH-tuples, which cost  $2\lambda$  in proof and in verification costs, see Section 4.2.

*Scalar-in-the-Head.* Since scalar operations on additively-shared secret values are linear, they can be computed locally by parties in MPC protocols. Similarly, for public  $c$  and secret  $a$ , an MPCitH prover can secret-share  $a = \sum_{i \in [n]} a_i$  and have every party compute  $[ca_i]E_{i-1}$  and pass it on to the next one in the same round-robin fashion as the multiplication protocol. This will clearly result in  $E_n = [ca]E_0$ . Furthermore, since the verifier sees that every MPC party used the public value  $c$ , there is no trusted helper required here.

**4.4.2 Exponentiation-in-the-Head with trusted setup.** Proving Exponentiation with MPCitH works along the same idea as multiplication. For exponentiation to the power  $e$ , we need tuples of the type

$$B = (\text{id}; (x_i^{(k)})_{i \in [n]}^{k \in [e]}, M_x) \in \{0, 1\}^* \times (\mathbb{Z}_N)^{e \cdot n} \times \mathcal{E}, \quad (8)$$

<sup>6</sup> Alternatively,  $n=4$ , leads to the higher average of  $2.00\lambda$  for the prover and the slightly lower average of  $2.50\lambda$  for the verifier.

MPC exponentiation protocol for  $P_i(\alpha, x_i^{(1)}, \dots, x_i^{(e)})$

---

```

1 : from  $P_{i-1}$  receive  $F_{i-1}$  ( $P_1$  receives  $F_0 = E_0$ )
2 : if  $i < n$  then
3 :    $F_i \leftarrow \left[ \sum_{k=1}^e \binom{e}{k} \alpha^{e-k} x_i^{(k)} \right] F_{i-1}$ 
4 :   send  $F_i$  to  $P_{i+1}$ 
5 : if  $i = n$  then
6 :    $F_n \leftarrow \left[ \alpha^e + \sum_{k=1}^e \binom{e}{k} \alpha^{e-k} x_i^{(k)} \right] F_{i-1}$ 
7 :   Broadcast( $F_n$ )

```

**Fig. 6.** MPC pseudocode for exponentiation

where  $M_x = [x]E_0$ . The shares are defined, such that  $\sum_{i=1}^n x_i^{(k)} = x^k$ . The trusted third party  $\mathcal{T}$  sends  $\text{View}_P(B) = (\text{id}; (x_i^{(k)})_{i \in [n]})$  to the prover and  $\text{View}_V(B) = (\text{id}; M_x)$  to the verifier. The prover distributes the values  $(x_i^{(k)})_{k \in [e]}$  to each party  $P_i$ , together with  $\alpha = a - x$ . We describe the MPC protocol in Figure 6, which we denote  $\text{ExplTH}_n^e(\alpha, B_P)$ . We defer the full interactive proof of knowledge to Appendix B.2. We note that, in contrast to the approach from Section 4.3 (found in Appendix B.1), this cost is independent of the exponent  $e$ .

**4.4.3 Polynomial evaluation in-the-Head with trusted setup.** The protocol from Figures 6 and 13 can be extended to the case where multiple parties want to evaluate a public polynomial on a shared input.

Let  $f(x) = \sum_{k=0}^d f_k x^k$ ; then we can see that  $[f(x)]E$  can be evaluated on a shared  $x$ , using the consecutive application of exponentiations. Thus, this can be achieved with the same trusted setup as in the case of exponentiation, see Equation (8). We summarize the MPC protocol in Figure 7. The full zero-knowledge protocol follows straightforwardly from the protocol in Figure 13.

**4.4.4 Removing the trusted helper with cut-and-choose.** We now show how the cut-and-choose technique can be used to remove the trusted helper in our multiplication and exponentiation protocols. The idea is that the prover now generates the structured tuple instead of the helper. However, the verifier must also be convinced that the elements in this tuple are well formed, such as e.g.  $z = xy$  or  $M_x = [x]E_0$ . This can be achieved with the cut-and-choose method described in Section 2.2: the prover precomputes and commits to a large amount  $m$  of these tuples and is then challenged to open  $m - \tau$  of them. Afterwards, the verifier can check that they have the desired structure.

The prover then runs the proof using the  $\tau$  undisclosed tuples, also convincing the verifier that the proof statement is correct. These proofs are computed by

MPC polynomial evaluation protocol for  $P_i(\alpha, f, x_i^{(1)}, \dots, x_i^{(d)})$

- 1 : from  $P_{i-1}$  receive  $F_{i-1}$  ( $P_1$  receives  $F_0 = E_0$ )
- 2 : **if**  $i < n$  **then**
- 3 :  $F_i \leftarrow \left[ \sum_{e=1}^d \sum_{k=1}^e f_k \binom{e}{k} \alpha^{e-k} x_i^{(k)} \right] F_{i-1}$
- 4 : **send**  $F_i$  to  $P_{i+1}$
- 5 : **if**  $i = n$  **then**
- 6 :  $F_n \leftarrow \left[ \left( f_0 + \sum_{e=1}^d \alpha^e \right) + \sum_{e=1}^d \sum_{k=1}^e \binom{e}{k} \alpha^{e-k} x_i^{(k)} \right] F_{i-1}$
- 7 : **Broadcast**( $F_n$ )

**Fig. 7.** MPC pseudocode for polynomial evaluation

running  $\text{MultiTH}_n$  or  $\text{ExplTH}_n^e$ , respectively, for each of the  $\tau$  tuples. If all of these checks succeed for appropriate choices of  $m$  and  $\tau$ , the verifier is convinced of the truth of the statement up to a negligible error probability.

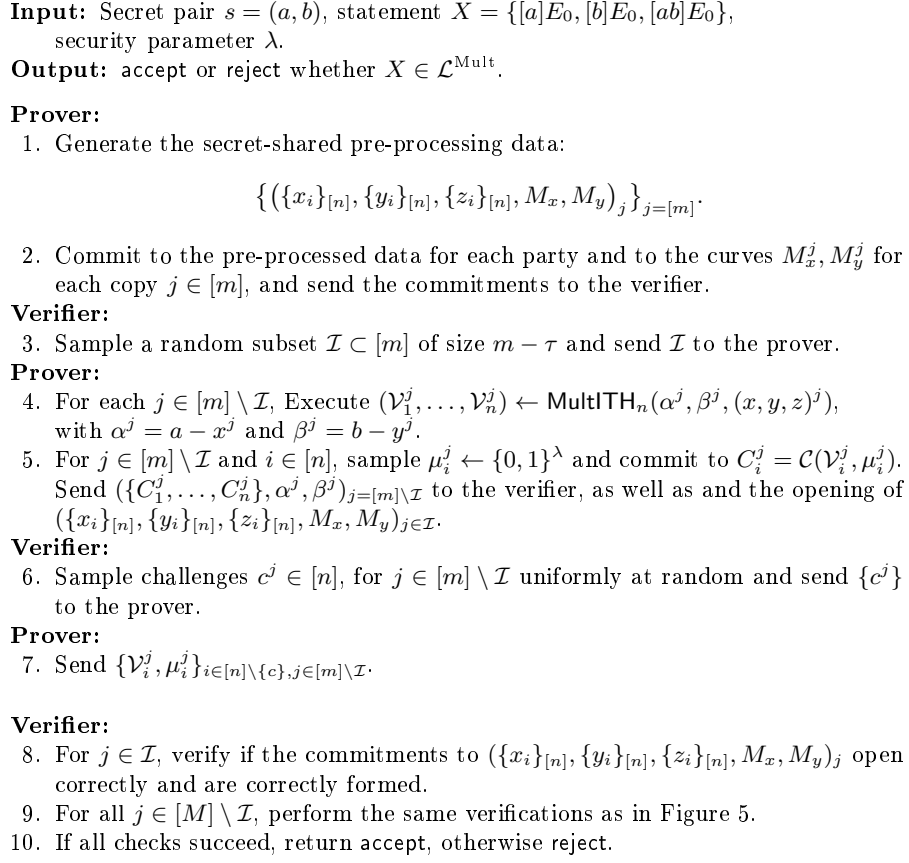
We present the protocol for  $\mathcal{L}^{\text{Mult}}$  in Figure 8. The protocol for  $\mathcal{L}^{\text{Exp}}$  can be built with the exact same tools from the protocol in the previous section and can be found in Appendix B.3.

**Theorem 4.** *Assuming ideal commitments, the protocol in Figure 8 is an interactive and honest-verifier zero-knowledge proof of knowledge for the language  $\mathcal{L}^{\text{Mult}}$  with soundness error  $\epsilon_{\text{cnc}}(m, n, \tau)$ .*

*Proof. Correctness:* It is clear that correctly formed tuples pass the verification conditions for  $j \in \mathcal{I}$ . For each execution  $j \in [m] \setminus \mathcal{I}$ , correctness follows from the correctness of  $\text{MultiTH}$ .

**Special soundness:** By the same argument as for Theorem 3, for a given  $j$ , the extractor can obtain  $x^j$  and  $y^j$  by using a malicious prover’s accepting responses to two different party challenges  $c^j$  and  $c'^j$ . By also rewinding the prover back to the commitment of the pre-processing data, and obtaining a third accepting transcript with a different opening of  $\tau$  datasets, the extractor can ensure that the  $(x^j, y^j, z^j)$  tuple used above is a valid multiplication tuple.

**Zero-knowledge:** To output an indistinguishable transcript, the simulator  $\mathcal{S}$  first samples  $\mathcal{I}$  at random and then generates honest secret-sharings  $(\{x_i\}_{[n]}, \{y_i\}_{[n]}, \{z_i\}_{[n]}, M_x, M_y)_j$  for  $j \in \mathcal{I}$ . For the remaining  $j \in [m] \setminus \mathcal{I}$ ,  $\mathcal{S}$  does as for Theorem 3 by sampling  $\alpha^j$  and  $\beta^j$  at random and setting  $M_x^j = [-\alpha^j]E_a$  and  $M_y^j = [-\beta^j]E_b$ . It then falsifies the round-robin computation of  $F_n^j$  in the same way: by sampling the challenge  $c^j$  at random, and computing  $F_{c-1}^j$  “forwards” from  $E_0$  and  $F_c^j$  “backwards” from  $E_{ab}$ . Since  $x_c^j, y_c^j, z_c^j$  remain hidden from the Verifier, and since there is no commitment as in the protocol with trusted setup, no other simulation is necessary.



**Fig. 8.** Interactive ZK proof for  $\mathcal{L}^{\text{Mult}}$  using MPCitH without trusted party.

*Correctness:* Here,  $F_n^j$  will equal the correct curves by construction and the views will be consistent due to the “forward” and “backward” computations.

*Indistinguishability:* The public values  $\alpha^j$  and  $\beta^j$  are sampled at random, which means they are distributed identically to the protocol, assuming that the Verifier has no knowledge of  $x^j, y^j, z^j$ . Similarly,  $M_x^j$  and  $M_y^j$  are distributed identically to an honest tuple. The idealised commitments perfectly hide  $x_c^j, y_c^j, z_c^j$  from the Verifier, so the perfect secrecy of the additive secret-sharing scheme implies the perfect zero-knowledge of the protocol.  $\square$

*Cost.* In the protocol of Figure 8, the prover computes a total of  $2m + n\tau$  isogenies,  $2m$  when generating the tuples and  $n\tau$  when running the MultITH protocol. The verifier checks  $2(m - \tau)$  of these tuples and verifies the protocol using another  $(n + 1)\tau$  isogenies, yielding the total  $2m + (n - 1)\tau$ .

The computational costs of the protocols introduced in this section depend on the choices of  $m$  and  $\tau$ , which themselves are determined by the equation established in Section 2.2. Using numerical techniques, we found that choosing  $n = 3$  always minimizes these costs. In particular we can find, that for the multiplication protocol, the prover has to approximately compute  $4.96\lambda$  isogenies and the verifier  $4.25\lambda$ .

**4.4.5 Cost overview.** We summarize the costs of the protocols introduced throughout this section and in Appendix B in Table 2.

**Table 2.** Number of isogeny computations of the prover and verifier for the protocols introduced in this work. The costs are given in terms of the security parameter  $\lambda$ , to ensure a soundness error of  $2^{-\lambda}$ . We further indicate, if the protocols need a trusted setup or not and which type of statements they are proving. We note that  $\text{Scal.Prove}_{k=1}$  coincides with the binary CSI-FiSh ID-protocol. The subscripts  $\mathcal{T}$  and  $cnc$  indicate the protocol versions with trusted setup and cut-and-choose, respectively. We also note that the cost for  $\text{ExplTH}_{\mathcal{T}}$  and  $\text{ExplTH}_{cnc}$  are the same as for the respective polynomial evaluation protocols described in Section 4.4.3.

	Prover cost	Verifier cost	Trusted Party	Statement type
$\text{Scal.Prove}_{k=1}$	$\lambda$	$\lambda$	No	$(E, [s]E)$
$\text{Scal.Prove}_{k=2}$	$2\lambda$	$2\lambda$	No	$(E, [a]E, [b]E, [a+b]E)$
$\text{Mult.Prove}$	$4\lambda$	$4\lambda$	Yes	$(E, [a]E, [b]E, [ab]E)$
$\text{MultITH}_{\mathcal{T}}$	$1.89\lambda$	$2.52\lambda$	Yes	
$\text{MultITH}_{cnc}$	$4.96\lambda$	$4.25\lambda$	No	
$\text{Exp.Prove}$	$\lceil \log_2 e \rceil 6\lambda$	$\lceil \log_2 e \rceil 6\lambda$	Yes	$(e, [a]E, [a^e]E)$
$\text{ExplTH}_{\mathcal{T}}$	$1.89\lambda$	$1.89\lambda$	Yes	
$\text{ExplTH}_{cnc}$	$3.52\lambda$	$3.52\lambda$	No	

## 4.5 New signatures

To give an idea of the applications we can build with the tools from this section, we introduce two examples of signature schemes, based on proofs of scalar multiplication and multiplication, respectively. The ideas behind these signature schemes are loosely based on the schemes of Boneh, Lynn and Shacham [18] (BLS) and of Zhang, Safavi-Naini and Susilo [62] (ZSS). The original schemes, proposed in the discrete logarithm setting, produce particularly short signatures and are verified using elliptic curve pairings, neither of which is the case here.

Rather, we present the signatures as instructive examples on how (scalar) multiplication proofs can be used to prove statements reminiscent of pairings in elliptic curve cryptography. The underlying observation is that statements of the language  $\mathcal{L}^{\text{Mult}}$ , e.g. tuples of the form  $(E, [a]E, [b]E, [ab]E)$  have a similar feel

<b>Keygen(pp)</b> 1. Sample $s \leftarrow \mathbb{Z}_N$ and compute $E_s = [s]E_0$ . 2. Return $(sk, pk) = (s, E_s)$ .	
<b>Scal-Sign(<math>m, s</math>)</b> 1. Compute $\sigma = [sH(m)]E_0$ . 2. Construct a $\mathcal{L}_2^{\text{Scal}}$ -proof $\pi$ for the statement $((E_0, E_s), (E_0, \sigma, H(m)))$ . 3. Return $\sigma, \pi$ .	<b>Mult-Sign(<math>m, s</math>)</b> 1. Compute $\sigma = \left[\frac{1}{s+H(m)}\right]E_0$ and $F = [H(m)]E_s$ . 2. Construct a $\mathcal{L}^{\text{Mult}}$ -proof $\pi$ for the statement $(E_0, \sigma, F, [1]E_0)$ . 3. Return $\sigma, \pi$ .
<b>Scal-Verify(<math>m, E_s, \sigma, \pi</math>)</b> 1. Compute $H(m)$ and verify $(H(m), E_s, \sigma)$ using $\pi$ . 2. If verification succeeds, return True, otherwise False.	<b>Mult-Verify(<math>m, E_s, \sigma, \pi</math>)</b> 1. Compute $F = [H(m)]E_s$ and verify $\sigma$ using $\pi$ . 2. If verification succeeds, return True, otherwise False.

**Fig. 9.** Two signature schemes using proofs of scalar multiplication or multiplication.

to pairing-based verification equations of the type  $e([a]P, [b]P) = e(P, [ab]P)$ . The caveats with respect to this interpretation are plentiful, however. As an example, elliptic curve points and the codomain of pairings both form groups, and operations in these groups are often crucial to allow verification, a perk that the elliptic curve set  $\mathcal{E}$  doesn't benefit from. Furthermore, while pairing equations can be performed using public elements, in our setting, we always need the prover to generate these publicly verifiable proofs first.

With this in mind, we present our signatures in Figure 9. The first scheme uses a proof for the language  $\mathcal{L}_2^{\text{Scal}}$ , while the latter uses proofs for  $\mathcal{L}^{\text{Mult}}$ . We note that any of the appropriate proof systems introduced in this work can be used. For both cases, we define the hash function  $H : \{0, 1\}^* \rightarrow \mathbb{Z}_N$ .

Security of these schemes immediately follows from the security of the proof schemes they employ. We note however, that in the multiplicative signature, for the element  $s + H(m)$  to be invertible, we would need to assume that  $N$  is prime, so that  $\mathbb{Z}_N$  is a field, since in any ring, the "allowed" values for  $H(m)$  would reveal information about  $s$ .

While the signatures in this section do not outperform the current state-of-the-art isogeny-based signature schemes (see Table 2 for reference), we hope that this inspires more research into other potential applications of the tools presented in this work.

## 5 An oblivious pseudo-random function

We finish our work by introducing an oblivious pseudo-random function based on the tools developed throughout the last sections.

An oblivious pseudo-random function (OPRF) is a protocol between a client  $C$  and a server  $S$ . The server has a secret key  $k$ , which defines a function  $F_k(\cdot)$ . The

client has a secret input  $m$ , on which it wants to evaluate this function. The goal of an OPRF is for the client to receive the evaluation  $F_k(m)$  without learning anything about  $k$ , while the server doesn't learn anything, i.e. neither the input  $m$ , nor the output  $F_k(m)$ . A *verifiable* OPRF further allows the client to verify that the server has indeed correctly used its secret  $k$  towards the computation of  $F_k(x)$ .

In our case, we assume the secret key of the server to be a polynomial  $f(x)$ , represented in terms of the polynomial coefficients  $k = (f_0, f_1, \dots, f_d)$ , where  $d = \deg f$ . On input a message  $m$ , we define the OPRF evaluation to be the function  $F_k(m) = [f(m)]E_0$ , for some starting curve  $E_0$ .

The idea behind the OPRF is that two parties engage in a polynomial evaluation protocol similar to the idea in Section 4.4.3. A major difference here, is that the polynomial coefficients are not public, but rather secrets of the server. As a result, they also have to be hidden. Therefore the parties jointly have to evaluate terms of the type  $[f_j m^j]E$  with  $f_j$  and  $m$  hidden. The resulting protocol is therefore rather a blend of the multiplication and exponentiation protocols introduced previously. To compute terms of this type, we assume that there is a functionality  $\mathcal{T}$  that generates the following kind of tuples:

$$\begin{aligned} B &= (\text{id}; x, \{y_j\}_{j \in [d]}, (\widetilde{z}_C, \widetilde{z}_S), \{(z_C^{(j,k)}, z_S^{(j,k)})\}_{k \in [j-1]}^{j \in [d]}), \\ \text{View}_C(B) &= (\text{id}; x, \widetilde{z}_C, \{z_C^{(j,k)}\}_{k \in [j-1]}^{j \in [d]}), \\ \text{View}_S(B) &= (\text{id}, \{y_j\}_{j \in [d]}, \widetilde{z}_S, \{z_S^{(j,k)}\}_{k \in [j-1]}^{j \in [d]}), \end{aligned} \tag{9}$$

such that<sup>7</sup>

$$z_C^{(j,k)} + z_S^{(j,k)} = y_j x^k \quad \text{and} \quad \widetilde{z}_C + \widetilde{z}_S = \sum_{j=1}^d y_j x^j.$$

We present our construction in Figure 10 and discuss its security in the theorem below. Some of the security properties of our OPRF will depend on properties of the polynomial  $f$ . We outline the necessary restrictions as part of the proof and defer the discussion about concrete instantiations of  $f$  to Section 5.1.

**Theorem 5.** *The protocol in Figure 10 satisfies the security requirements of an OPRF, i.e. correctness, hiding against a malicious client, hiding against a malicious server, binding and one-more unpredictability.*

*Proof.* We omit the full definition of the security properties of an OPRF here, and refer the reader to [17, 38, 55] more details.

<sup>7</sup> Note that we could as well have defined the tuples to contain  $\{(z_C^{(j,k)}, z_S^{(j,k)})\}_{k \in [j]}$ . Since in our protocol, the coefficients of terms where  $j = k$  is always 1, we can summarize all of these terms in  $\widetilde{z}_C$  and  $\widetilde{z}_S$ , which leads to a smaller trusted setup by reducing the amount of such terms by  $d - 1$ .



**Input:** Secret key  $k = (f_0, f_1, \dots, f_d) \in \mathbb{Z}_N^{d+1}$  held by the server  $S$ , secret input  $m \in \mathbb{Z}_N^*$  by the client  $C$ . Shared tuple  $B$  as in equation (9).

**Output:** Evaluation  $F_k(m) = [f(m)]E_0$ .

**Client:**

1. Compute  $\alpha = m - x$  and send it to the server.

**Server:**

2. For  $j = 1, \dots, d$ , compute  $\beta_j = f_j - y_j$ .
3. Compute

$$E_S = \left[ f_0 + \widetilde{z}_S + \sum_{j=1}^d f_j \alpha^j + \sum_{j=1}^d \sum_{k=1}^{j-1} \binom{j}{k} \alpha^{j-k} z_S^{(j,k)} \right] E_0.$$

4. Send  $(\beta_1, \dots, \beta_d)$  and  $E_S$  it to the client.

**Client:**

5. Return

$$\left[ \widetilde{z}_C + \sum_{j=1}^d \beta_j x^j + \sum_{j=1}^d \sum_{k=1}^{j-1} \binom{j}{k} \alpha^{j-k} (\beta_j x^k + z_C^{(j,k)}) \right] E_S.$$

**Fig. 10.** Oblivious pseudo-random function based on joint polynomial evaluation.

**Correctness:** By summing the action of the client and of the server, it can be quickly verified that the action on  $E_0$  is given by

$$f_0 + \widetilde{z}_S + \widetilde{z}_C + \sum_{j=1}^d (f_j \alpha^j + \beta_j x^j) + \sum_{j=1}^d \sum_{k=1}^{j-1} \binom{j}{k} \alpha^{j-k} (z_S^{(j,k)} + z_C^{(j,k)} + \beta_j x^k).$$

By plugging in the definitions of Equation (9) as well as the fact that  $\beta_j + y_j = f_j$ , this simplifies to

$$f_0 + \sum_{j=1}^d f_j \left( \alpha^j + \sum_{k=1}^{j-1} \binom{j}{k} \alpha^{j-k} x^k + x^j \right),$$

which is the expression  $\sum_{j=0}^d f_j (\alpha + x)^j = f(m)$ .

**Hiding against malicious server:** It is clear that  $\alpha$  information-theoretically hides  $m$ : as the functionality  $\mathcal{T}$ , the simulator samples  $B$  uniformly at random from the different sets and sends  $\text{View}_S(B)$  to the server. Then, as the client, it samples  $\alpha \leftarrow \mathbb{Z}_N$  and send it to the server. This simulation is perfectly indistinguishable from the real execution of the protocol.

**Hiding against malicious client:** To prove this, we show that the malicious client has a negligible advantage in the hiding game. After receiving  $\alpha$  from the adversary, the challenger samples  $b \leftarrow \{0, 1\}$ . Then,

– if  $b = 0$ , it computes  $\beta_j^{(0)} = f_j - y_j$  for  $j = 1, \dots, d$  and

$$r^{(0)} = f_0 + \widetilde{z}_S + \sum_{j=1}^d f_j \alpha^j + \sum_{j=1}^d \sum_{k=1}^{j-1} \binom{j}{k} \alpha^{j-k} z_S^{(e,k)},$$

– if  $b = 1$ , it samples  $r^{(1)} \leftarrow \mathbb{Z}_N$  and  $\beta_j^{(1)} \leftarrow \mathbb{Z}_N$  for  $j = 1, \dots, d$ .

In either case, the server sends  $(\beta_1^{(b)}, \dots, \beta_d^{(b)})$  and  $E_S^{(b)} = [r^{(b)}]E_0$  to the client. The adversary then outputs  $b' \in \{0, 1\}$  and wins the game if  $b' = b$ . It is clear that  $(\beta_1^{(1)}, \dots, \beta_d^{(1)})$  and  $(\beta_1^{(0)}, \dots, \beta_d^{(0)})$  follow perfectly indistinguishable distributions as  $f_j$  and  $y_j$  are unknown to the adversary. Similarly,  $r^{(0)}$  and  $r^{(1)}$  should be indistinguishable, but we note, that in the first case, the adversary can evaluate the OPRF correctly and receive  $[f(m)]E_0$  after applying its half of the action. Now, due to the freeness of the group action, the adversary will still not be able to distinguish both cases, as long as the output distribution of the polynomial  $f(m)$  is indistinguishable from uniform. We note that this is an important restriction when we are working in rings. In particular, since the client's queries are hidden, a malicious client could only send inputs in a subgroup of  $\mathbb{Z}_N$ , which it could easily distinguish from a random  $r^{(1)}$ . We therefore have to assume that we are working in a subgroup of the class group of prime order if we want to protect against malicious clients. For semi-honest clients, we emphasize that this restriction is not necessary.

**Binding:** Due to the freeness of the group action, we only have a collision, if  $f(m) = f(m')$ . The OPRF is binding, if  $f$  is collision-resistant.

**One-more unpredictability:** We assume the adversary  $\mathcal{A}$  has oracle access to an OPRF oracle, which on input  $m$  outputs  $[f(m)]E_0$ . After  $r$  evaluations of the OPRF, the adversary has knowledge of

$$(m_1, [f(m_1)]E_0), \dots, (m_r, [f(m_r)]E_0).$$

The client breaks one-more security by finding a pair  $m^*, [f(m^*)]E_0$ , where  $m^* \notin \{m_1, \dots, m_r\}$ . The hardness of this again depends on the polynomial  $f$ . We discuss these restrictions in Section 5.1.  $\square$

## 5.1 Choosing the polynomial

The first restriction from Theorem 5, is that we have to work in a prime order subgroup of the class group to guarantee security against a malicious adversary. So, throughout this section, we assume that we are working in a prime order subgroup of the class group and that  $\mathbb{Z}_N$  constitutes a field. In the proof of Theorem 5, we also found the following restrictions on  $f$ :

1.  $f$  has output distribution indistinguishable from uniform,
2.  $f$  is collision-resistant,
3. the one-more unpredictability problem is hard.

From Equation (9), we see that polynomials of high degree lead to a higher communication and storage cost in terms of tuples. We can easily count that the number of elements in the tuple  $B$  is  $3 + d^2$ . As a further restriction, we therefore add, that

4.  $f$  has small degree.

It is immediately clear that the first two restrictions can be achieved by permutation polynomials, i.e. a bijective polynomial  $f : \mathbb{Z}_N \rightarrow \mathbb{Z}_N$ . Any linear polynomial already fits the bill here, but unfortunately, linear polynomials turn out to not be secure against quantum adversaries. If  $d = 1$ , the output is a multiplication with an offset, i.e.  $[f_0 + f_1 m]E_0$ . In the reduction from GAIP to Parallelization outlined by Galbraith et al. [39], the authors show that access to an oracle, which on input  $m$  outputs  $[f_1 m]E$  is sufficient to recover  $f_1$ . It is clear that this breaks one-more-unpredictability.

For  $d = 2$ , permutation polynomials only exist over fields with characteristic 2, which contradicts the premise that  $N$  is a prime. Thankfully, when going to  $d = 3$ , if  $p \equiv 2 \pmod{3}$ , any polynomial of the form  $f(m) = a(m+b)^3 + c$  with  $a \neq 0$  is a permutation polynomial [31]. We do however note that heuristically, a polynomial of degree 2 (with non-zero coefficients) is also enough. In a field, using a quadratic polynomial satisfies the binding property, since for every  $f(m)$ , there exists *at most* a second input that evaluates to  $f(m)$  (this follows from the fact that  $f(m) = f(m')$  is an equation of degree 2 and therefore admits at most two solutions). Similarly, since at least half of  $\mathbb{Z}_N$  is covered by the outputs of  $f$  (with each element reached at most twice), we can also guarantee the hiding property against a malicious client.

So, we have found polynomials that fit our bill, assuming the one-more unpredictability problem is hard. Let us first rephrase the latter as a game, where the adversary  $\mathcal{A}$  against one-more unpredictability has oracle access to an OPRF oracle  $\mathcal{O}$ , which on input  $m$ , outputs  $[f(m)]E_0$ . After polynomially many queries,  $\mathcal{A}$  outputs  $(m^*, E^*)$  with  $m^*$  not previously queried to  $\mathcal{O}$ , and wins the game, if  $E^* = [f(m^*)]E_0$ . We first start by noticing that our assumption reduces to (Scalar-)CDH, i.e. access to such an oracle, allows us to find  $[f(m^*)]E_0$  as follows (for simplicity, we outline the case  $d = 2$ , while other cases work analogously):

1. Query  $\mathcal{O}$  on  $m \in \{-1, 0, 1\}$  to get  $[f_0]E_0$ ,  $[f_0 + f_1 + f_2]E_0$  and  $[f_0 - f_1 + f_2]E_0$ .
2. Use the CDH-oracle to build  $[f_1 + f_2]E_0$  and  $[-f_1 + f_2]E_0$  from these by subtraction of  $[f_0]E_0$ , then build  $[2f_1]E_0$  and  $[2f_2]E_0$  from addition and subtraction.
3. For any message  $m^*$ , compute  $m^*/2$  and  $(m^*)^2/2$  and call the Scalar-CDH oracle to compute  $[f_1 m^*]E_0$  and  $[f_2 (m^*)^2]E_0$ .
4. Finally, compute and output  $[f_0 + f_1 m^* + f_2 (m^*)^2]E_0 = [f(m^*)]E_0$ .

Showing the converse is less trivial, however. Assuming oracle access to an OPRF-oracle, which on input  $m$  gives us the output  $[f(m)]E_0$  does not give us much to work with, when we want to use this to break some assumption. As a result, we have to rely on a more heuristic argumentation to convince ourselves that it is hard for  $\mathcal{A}$  to win the game. To this end, let's view our map as follows.

$$\begin{array}{ccccc} \mathbb{Z}_N & \longrightarrow & \mathbb{Z}_N & \longrightarrow & \mathcal{E} \\ m & \longmapsto & f(m) & \longmapsto & [f(m)]E_0 \end{array}$$

The adversary interacting with  $\mathcal{O}$  chooses  $m$  and learns  $[f(m)]E_0$ , thus the left and the right hand side of our map. We can argue that from both of these sides, the adversary is not able to infer any information about  $f(m)$ . We already know from the proof of Theorem 5 that by choosing  $m$ , we can't learn anything about  $f(m)$  as long as it has output indistinguishable from uniform. Furthermore, inferring anything about  $f(m)$  from  $[f(m)]E_0$  contradicts the assumptions of a cryptographic group action being one-way and unpredictable. This last sentence has to be taken with a large grain of salt, as this argument would also apply in the case where  $f(m)$  is a linear function, which we have proven to be insufficient, and allowing the extraction of the secret. However, linear functions define instances of the hidden subgroup problem, while non-linear (non-monomial) polynomials ostensibly do not. As a result, none of the standard attacks from the literature seem to be applicable to our problem, from which we conjecture that it is a hard problem. With this, we leave further scrutiny of our security assumption as an open problem for future research.

## 5.2 Adding verifiability

We discuss how to turn our OPRF into a verifiable OPRF. The idea is similar to protocols like the one in Figures 4 and 11, where we add elliptic curves to the honestly generated tuples  $B$  and use `Scal.Prove` in order to convince the verifier that the computation has been done as instructed.

To this end, we define the server's public key as  $\{P_j = [f_j]E_0\}_{j=0,\dots,d}$  and we need to add the following publicly visible elements to the trusted tuple

$$\widetilde{M} = [\widetilde{z}_S]E_0, \{M^{(j,k)} = [z_S^{(j,k)}]E_0\}_{\substack{j \in [d] \\ k \in [j-1]}}.$$

Then the server computes the different additions in Step 3 of Figure 10 consecutively, starting from its public key, and appends a proof of correct computation. The server steps then become

$$\left[ \alpha^j f_j \right]_{j \in [d]} \left[ \binom{j}{k} \alpha^{j-k} z_S^{(j,k)} \right]_{k \in [j-1]}^{j \in [d]} \left[ \widetilde{z}_S \right] P_0,$$

with appended proofs at each step, that the correct witness was used with the correct factor from a curve  $E_{i-1}$  to  $E_i$ , e.g.

$$\text{Scal.Prove}_\lambda \left( z_S^{(j,k)}, \{(E_0, M^{(j,k)}), (E_{i-1}, E_i, \binom{j}{k} \alpha^{j-k})\}, \right).$$

All the proofs are sent to the client, which verifies them and finally computes its own action, to get the OPRF output  $[f(m)]E_0$ .

### 5.3 Comparison to the literature

We end this section by comparing the cost of our OPRF with CSIDH-based (and other isogeny-based) OPRFs in the literature. The first CSIDH-based OPRF was introduced by Boneh et al. [17] and is based on a Naor-Reingold PRF, where the server and the client engage in  $\lambda \binom{2}{1}$ -oblivious transfers. Note that this initial design is based on the OT by Lai et al. [51] which requires a trusted setup. Follow-up work by Heimberger et al. [42] improves the protocol of Boneh et al. by reducing its round-complexity and presents a new OPRF protocol without trusted setup, and which also works when the class group is unknown. We summarize the different protocols and their relative costs in Table 3, and compare polynomial degrees  $d = 2$  and  $3$  of our design.

In our protocol, a malicious client comes at no extra communication or computational cost to the semi-honest case and our number of rounds is optimal, even in the verifiable case. The non-verifiable version of our protocol strongly outperforms the other protocols from the literature in terms of computation and communication costs, by at least two orders of magnitude. Even when we add verifiability, which hasn't been done before in the CSIDH setting, our protocol still outperforms the designs of Boneh et al. [17] and only has about twice the computational cost of the semi-honest protocol by Heimberger et al. [42], while still being round-optimal and having lower communication cost.

*Remark 2.* For completeness, we would also like to note that an isogeny-based OPRF has recently been proposed by Basso [8] in the M-SIDH (masked torsion point) setting of Fouotsa, Moriya and Petit [36]. Basso's work introduces a round-optimal verifiable OPRF based on a trusted setup, with a total communication cost of approximately  $60\lambda \log p + 87\lambda^2$ . Due to the point masking procedure of M-SIDH, the prime is chosen to be 8868 bits long to achieve NIST level I security. A comparison with our protocol in terms of theoretical costs is not directly possible, as Basso's works with isogenies over  $\mathbb{F}_{p^2}$  and with operations such as scalar multiplication and pushing through of points. The author foregoes a direct analysis of the computational complexity. Furthermore, the quantum security of CSIDH in relation to NIST level I is not completely settled yet [19, 57], so that it is hard to define the same security level in both settings. However, we note that even with the most conservative estimate of a 4096-bit prime for level I security, the communication cost of our OPRF outperforms Basso's by a factor 8 in the verifiable case (4.6, if  $d = 3$ ), and by a factor 122 in the non-verifiable case (81, if  $d = 3$ ). We point the interested reader to [42, Section 8] for a more thorough comparison of current post-quantum OPRF designs and to [20] for a more general overview of OPRF designs.

### 5.4 Removing the trusted setup

Generating trusted tuples of the form required by our protocol can be seen as an arithmetic computation over  $\mathbb{Z}_N$ . When  $\mathbb{Z}_N$  is a field, this can be efficiently realized, even with malicious security, using multi-party protocols. In particular,

**Table 3.** Comparison of the OPRFs by Boneh et al. [17] and Heimberger et al. [42] with our protocol in Figure 10 and the verifiable OPRF from Section 5.2, both for the polynomial degrees  $d = 2, 3$ . The computational cost is expressed in the amount of group actions to be performed. \*In Section 5.4, we discuss how the trusted setup can be removed for our protocol. We note that this increases the total computational and communication costs as outlined in that section.

Source	Malicious Client	Verifiable	Number of Rounds	Total Computational cost	Total Communication cost	No Trusted Setup	Without Class Group
[42]	×	×	$2\lambda + 2$	$3\lambda + 3$	$(3\lambda + 2) \log p$	✓	✓
[17]	×	×	2	$5\lambda + 2$	$(2\lambda + 1) \log p + 2\lambda^2$	×	×
	✓	×	4	$11\lambda + 2$	$(5\lambda + 1) \log p + 5\lambda^2$	×	×
$d = 2$	✓	×	2	2	$6 \log p$	×	×
$d = 3$	✓	×	2	2	$9 \log p$	×	×
$d = 2$	✓	✓	2	$8\lambda + 4$	$(2\lambda + \frac{17}{2}) \log p + 4\lambda$	×	×
$d = 3$	✓	✓	2	$14\lambda + 7$	$(\frac{7}{2}\lambda + \frac{29}{2}) \log p + 7\lambda$	×	×

the MASCOT protocol of Keller, Orsini and Scholl [49] enables this, using only oblivious transfer (OT) as a public-key primitive. The particular advantage of using OT as the fundamental primitive is that, with symmetric-key OT extension techniques, a large number of tuples can be produced using a small number of OT instances [43, 48]. Furthermore, isogeny-based OT constructions exist [3, 6, 51].

OPRF is a useful tool for larger protocols, such as private set intersection; see Rindal and Schoppmann [58, Section 4] for the standard construction of a PSI protocol from an OPRF. Application to PSI means that larger numbers of OPRF calls are required. In the case of private contact discovery, one party may need to make hundreds of thousands of OPRF calls. In this scenario, the amortization offered by MASCOT brings a tremendous advantage to reduce the number of base OT executions that are necessary. For 128-bit fields, the MASCOT protocol for two parties, with full malicious security, can reach throughputs of up to 4,800 triples per second over a 1 Gbit/s network [49]. To this throughput cost we must add a one-time setup cost to execute the base-OTs required for the OT extension. Keller, Orsini and Scholl’s extension protocol requires  $\lambda$  base OTs [48] and Badrinarayanan et al.’s isogeny-based OT protocol requires 5 isogeny computations [6], thus totalling a one-time cost of  $5\lambda$  isogeny computations.

After the setup, each triple generated by the symmetric OT extension protocol is sufficient to construct one multiplication tuple for our trusted setups. When  $d = 2$ , we require four multiplications to construct the tuple of Equation (9) which implies that throughputs upwards of 1,000 tuples per second could be achieved. For 256-bit fields, we estimate that this would be reduced by a factor of 1/4, yielding a throughput of 250–300 tuples per second, and for  $d = 3$  still about half of that.

**Acknowledgments.** The authors would like to thank Karim Baghery, Steven Galbraith, Yi-Fu Lai, Emmanuela Orsini, Nigel Smart and Frederik Vercauteren for helpful discussions regarding the contents of this work. 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 ISOCRYPT - No. 101020788) and by CyberSecurity Research Flanders with reference number VR20192203. Cyprien Delpech de Saint Guilhem is a Junior FWO Postdoctoral Fellow under project 1266123N.

## References

1. Abdalla, M., Eisenhofer, T., Kiltz, E., Kunzweiler, S., Riepel, D.: Password-authenticated key exchange from group actions. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part II. LNCS, vol. 13508, pp. 699–728. Springer, Heidelberg (Aug 2022). [https://doi.org/10.1007/978-3-031-15979-4\\_24](https://doi.org/10.1007/978-3-031-15979-4_24)
2. Alamati, N., De Feo, L., Montgomery, H., Patranabis, S.: Cryptographic group actions and applications. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part II. LNCS, vol. 12492, pp. 411–439. Springer, Heidelberg (Dec 2020). [https://doi.org/10.1007/978-3-030-64834-3\\_14](https://doi.org/10.1007/978-3-030-64834-3_14)
3. Alamati, N., Montgomery, H., Patranabis, S., Sarkar, P.: Two-round adaptively secure MPC from isogenies, LPN, or CDH. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part II. LNCS, vol. 13091, pp. 305–334. Springer, Heidelberg (Dec 2021). [https://doi.org/10.1007/978-3-030-92075-3\\_11](https://doi.org/10.1007/978-3-030-92075-3_11)
4. Atapoor, S., Baghery, K., Cozzo, D., Pedersen, R.: CSI-SharK: CSI-FiSh with sharing-friendly keys. In: Simpson, L., Bae, M.A.R. (eds.) ACISP 2023. Lecture Notes in Computer Science, vol. 13915, pp. 471–502. Springer (2023). [https://doi.org/10.1007/978-3-031-35486-1\\_21](https://doi.org/10.1007/978-3-031-35486-1_21)
5. Atapoor, S., Baghery, K., Cozzo, D., Pedersen, R.: Practical robust DKG protocols for CSIDH. In: Tibouchi, M., Wang, X. (eds.) ACNS 2023, Part II. Lecture Notes in Computer Science, vol. 13906, pp. 219–247. Springer (2023). [https://doi.org/10.1007/978-3-031-33491-7\\_9](https://doi.org/10.1007/978-3-031-33491-7_9)
6. Badrinarayanan, S., Masny, D., Mukherjee, P., Patranabis, S., Raghuraman, S., Sarkar, P.: Round-optimal oblivious transfer and MPC from computational CSIDH. In: Boldyreva, A., Kolesnikov, V. (eds.) PKC 2023, Part I. LNCS, vol. 13940, pp. 376–405. Springer, Heidelberg (May 2023). [https://doi.org/10.1007/978-3-031-31368-4\\_14](https://doi.org/10.1007/978-3-031-31368-4_14)
7. Baghery, K., Cozzo, D., Pedersen, R.: An isogeny-based ID protocol using structured public keys. In: Paterson, M.B. (ed.) 18th IMA International Conference on Cryptography and Coding. LNCS, vol. 13129, pp. 179–197. Springer, Heidelberg (Dec 2021). [https://doi.org/10.1007/978-3-030-92641-0\\_9](https://doi.org/10.1007/978-3-030-92641-0_9)
8. Basso, A.: A post-quantum round-optimal oblivious PRF from isogenies. Cryptology ePrint Archive, Report 2023/225 (2023), <https://eprint.iacr.org/2023/225>
9. Basso, A., Kutas, P., Merz, S.P., Petit, C., Sanso, A.: Cryptanalysis of an oblivious PRF from supersingular isogenies. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part I. LNCS, vol. 13090, pp. 160–184. Springer, Heidelberg (Dec 2021). [https://doi.org/10.1007/978-3-030-92062-3\\_6](https://doi.org/10.1007/978-3-030-92062-3_6)
10. Baum, C., Delpech de Saint Guilhem, C., Kales, D., Orsini, E., Scholl, P., Zaverucha, G.: Banquet: Short and fast signatures from AES. In: Garay, J. (ed.)

- PKC 2021, Part I. LNCS, vol. 12710, pp. 266–297. Springer, Heidelberg (May 2021). [https://doi.org/10.1007/978-3-030-75245-3\\_11](https://doi.org/10.1007/978-3-030-75245-3_11)
11. Baum, C., Nof, A.: Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020, Part I. LNCS, vol. 12110, pp. 495–526. Springer, Heidelberg (May 2020). [https://doi.org/10.1007/978-3-030-45374-9\\_17](https://doi.org/10.1007/978-3-030-45374-9_17)
  12. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Feigenbaum, J. (ed.) CRYPTO'91. LNCS, vol. 576, pp. 420–432. Springer, Heidelberg (Aug 1992). [https://doi.org/10.1007/3-540-46766-1\\_34](https://doi.org/10.1007/3-540-46766-1_34)
  13. Beullens, W., Disson, L., Pedersen, R., Vercauteren, F.: CSI-RASh: Distributed key generation for CSIDH. In: Cheon, J.H., Tillich, J.P. (eds.) Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021. pp. 257–276. Springer, Heidelberg (2021). [https://doi.org/10.1007/978-3-030-81293-5\\_14](https://doi.org/10.1007/978-3-030-81293-5_14)
  14. Beullens, W., Dobson, S., Katsumata, S., Lai, Y.F., Pintore, F.: Group signatures and more from isogenies and lattices: Generic, simple, and efficient. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 95–126. Springer, Heidelberg (May / Jun 2022). [https://doi.org/10.1007/978-3-031-07085-3\\_4](https://doi.org/10.1007/978-3-031-07085-3_4)
  15. Beullens, W., Katsumata, S., Pintore, F.: Calamari and Falaff: Logarithmic (linkable) ring signatures from isogenies and lattices. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part II. LNCS, vol. 12492, pp. 464–492. Springer, Heidelberg (Dec 2020). [https://doi.org/10.1007/978-3-030-64834-3\\_16](https://doi.org/10.1007/978-3-030-64834-3_16)
  16. Beullens, W., Kleinjung, T., Vercauteren, F.: CSI-FiSh: Efficient isogeny based signatures through class group computations. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019, Part I. LNCS, vol. 11921, pp. 227–247. Springer, Heidelberg (Dec 2019). [https://doi.org/10.1007/978-3-030-34578-5\\_9](https://doi.org/10.1007/978-3-030-34578-5_9)
  17. Boneh, D., Kogan, D., Woo, K.: Oblivious pseudorandom functions from isogenies. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part II. LNCS, vol. 12492, pp. 520–550. Springer, Heidelberg (Dec 2020). [https://doi.org/10.1007/978-3-030-64834-3\\_18](https://doi.org/10.1007/978-3-030-64834-3_18)
  18. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. *Journal of Cryptology* **17**(4), 297–319 (Sep 2004). <https://doi.org/10.1007/s00145-004-0314-9>
  19. Bonnetain, X., Schrottenloher, A.: Quantum security analysis of CSIDH. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part II. LNCS, vol. 12106, pp. 493–522. Springer, Heidelberg (May 2020). [https://doi.org/10.1007/978-3-030-45724-2\\_17](https://doi.org/10.1007/978-3-030-45724-2_17)
  20. Casacuberta, S., Hesse, J., Lehmann, A.: Sok: Oblivious pseudorandom functions. In: 7th IEEE European Symposium on Security and Privacy, EuroS&P 2022, Genoa, Italy, June 6-10, 2022. pp. 625–646. IEEE (2022). <https://doi.org/10.1109/EUROSP53844.2022.00045>, <https://doi.org/10.1109/EuroSP53844.2022.00045>
  21. Castryck, W., Decru, T.: An efficient key recovery attack on SIDH. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part V. LNCS, vol. 14008, pp. 423–447. Springer, Heidelberg (Apr 2023). [https://doi.org/10.1007/978-3-031-30589-4\\_15](https://doi.org/10.1007/978-3-031-30589-4_15)
  22. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: An efficient post-quantum commutative group action. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part III. LNCS, vol. 11274, pp. 395–427. Springer, Heidelberg (Dec 2018). [https://doi.org/10.1007/978-3-030-03332-3\\_15](https://doi.org/10.1007/978-3-030-03332-3_15)



23. Chung, K.M., Hsieh, Y.C., Huang, M.Y., Huang, Y.H., Lange, T., Yang, B.Y.: Group signatures and accountable ring signatures from isogeny-based assumptions. Cryptology ePrint Archive, Report 2021/1368 (2021), <https://eprint.iacr.org/2021/1368>
24. Cohen, H., Lenstra Jr, H.W.: Heuristics on class groups of number fields. In: Number Theory Noordwijkerhout 1983: Proceedings of the Journées Arithmétiques held at Noordwijkerhout, The Netherlands July 11–15, 1983, pp. 33–62. Springer (2006)
25. Couveignes, J.M.: Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291 (2006), <https://eprint.iacr.org/2006/291>
26. Cozzo, D., Smart, N.P.: Sashimi: Cutting up CSI-FiSh secret keys to produce an actively secure distributed signing protocol. In: Ding, J., Tillich, J.P. (eds.) Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020. pp. 169–186. Springer, Heidelberg (2020). [https://doi.org/10.1007/978-3-030-44223-1\\_10](https://doi.org/10.1007/978-3-030-44223-1_10)
27. De Feo, L., Fouotsa, T.B., Kutas, P., Leroux, A., Merz, S.P., Panny, L., Wesolowski, B.: SCALLOP: Scaling the CSI-FiSh. In: Boldyreva, A., Kolesnikov, V. (eds.) PKC 2023, Part I. LNCS, vol. 13940, pp. 345–375. Springer, Heidelberg (May 2023). [https://doi.org/10.1007/978-3-031-31368-4\\_13](https://doi.org/10.1007/978-3-031-31368-4_13)
28. De Feo, L., Galbraith, S.D.: SeaSign: Compact isogeny signatures from class group actions. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part III. LNCS, vol. 11478, pp. 759–789. Springer, Heidelberg (May 2019). [https://doi.org/10.1007/978-3-030-17659-4\\_26](https://doi.org/10.1007/978-3-030-17659-4_26)
29. De Feo, L., Kohel, D., Leroux, A., Petit, C., Wesolowski, B.: SQISign: Compact post-quantum signatures from quaternions and isogenies. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part I. LNCS, vol. 12491, pp. 64–93. Springer, Heidelberg (Dec 2020). [https://doi.org/10.1007/978-3-030-64837-4\\_3](https://doi.org/10.1007/978-3-030-64837-4_3)
30. De Feo, L., Meyer, M.: Threshold schemes from isogeny assumptions. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020, Part II. LNCS, vol. 12111, pp. 187–212. Springer, Heidelberg (May 2020). [https://doi.org/10.1007/978-3-030-45388-6\\_7](https://doi.org/10.1007/978-3-030-45388-6_7)
31. Dickson, L.E.: Linear groups: With an exposition of the Galois field theory, vol. 6. BG Teubner (1901)
32. Eaton, E., Jao, D., Komlo, C., Mokrani, Y.: Towards post-quantum key-updatable public-key encryption via supersingular isogenies. In: AlTawy, R., Hülsing, A. (eds.) SAC 2021. LNCS, vol. 13203, pp. 461–482. Springer, Heidelberg (Sep / Oct 2022). [https://doi.org/10.1007/978-3-030-99277-4\\_22](https://doi.org/10.1007/978-3-030-99277-4_22)
33. El Kaafarani, A., Katsumata, S., Pintore, F.: Lossy CSI-FiSh: Efficient signature scheme with tight reduction to decisional CSIDH-512. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020, Part II. LNCS, vol. 12111, pp. 157–186. Springer, Heidelberg (May 2020). [https://doi.org/10.1007/978-3-030-45388-6\\_6](https://doi.org/10.1007/978-3-030-45388-6_6)
34. Felderhoff, J.: Hard homogenous spaces and commutative supersingular isogeny based diffie-hellman. Internship Report, LIX, Ecole polytechnique; ENS de Lyon (2019)
35. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO'86. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (Aug 1987). [https://doi.org/10.1007/3-540-47721-7\\_12](https://doi.org/10.1007/3-540-47721-7_12)
36. Fouotsa, T.B., Moriya, T., Petit, C.: M-SIDH and MD-SIDH: Countering SIDH attacks by masking information. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023,

- Part V. LNCS, vol. 14008, pp. 282–309. Springer, Heidelberg (Apr 2023). [https://doi.org/10.1007/978-3-031-30589-4\\_10](https://doi.org/10.1007/978-3-031-30589-4_10)
37. Fouotsa, T.B., Petit, C.: SimS: A simplification of SiGamal. In: Cheon, J.H., Tillich, J.P. (eds.) Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021. pp. 277–295. Springer, Heidelberg (2021). [https://doi.org/10.1007/978-3-030-81293-5\\_15](https://doi.org/10.1007/978-3-030-81293-5_15)
  38. Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword search and oblivious pseudorandom functions. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 303–324. Springer, Heidelberg (Feb 2005). [https://doi.org/10.1007/978-3-540-30576-7\\_17](https://doi.org/10.1007/978-3-540-30576-7_17)
  39. Galbraith, S., Panny, L., Smith, B., Vercauteren, F.: Quantum Equivalence of the DLP and CDHP for Group Actions. *Mathematical Cryptology* **1**(1), 40–44 (2021)
  40. Goldreich, O.: On expected probabilistic polynomial-time adversaries: A suggestion for restricted definitions and their benefits. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 174–193. Springer, Heidelberg (Feb 2007). [https://doi.org/10.1007/978-3-540-70936-7\\_10](https://doi.org/10.1007/978-3-540-70936-7_10)
  41. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. *SIAM Journal on Computing* **18**(1), 186–208 (1989)
  42. Heimberger, L., Meisinger, F., Hennerbichler, T., Ramacher, S., Rechberger, C.: OPRFs from Isogenies: Designs and Analysis. *Cryptology ePrint Archive, Paper 2023/639* (2023), <https://eprint.iacr.org/2023/639>
  43. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (Aug 2003). [https://doi.org/10.1007/978-3-540-45146-4\\_9](https://doi.org/10.1007/978-3-540-45146-4_9)
  44. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge proofs from secure multiparty computation. *SIAM J. Comput.* **39**(3), 1121–1152 (2009). <https://doi.org/10.1137/080725398>
  45. Joux, A.: MPC in the head for isomorphisms and group actions. *Cryptology ePrint Archive, Paper 2023/664* (2023), <https://eprint.iacr.org/2023/664>
  46. Kales, D., Zaverucha, G.: An attack on some signature schemes constructed from five-pass identification schemes. In: Krenn, S., Shulman, H., Vaudenay, S. (eds.) CANS 20. LNCS, vol. 12579, pp. 3–22. Springer, Heidelberg (Dec 2020). [https://doi.org/10.1007/978-3-030-65411-5\\_1](https://doi.org/10.1007/978-3-030-65411-5_1)
  47. Katz, J., Kolesnikov, V., Wang, X.: Improved non-interactive zero knowledge with applications to post-quantum signatures. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 525–537. ACM Press (Oct 2018). <https://doi.org/10.1145/3243734.3243805>
  48. Keller, M., Orsini, E., Scholl, P.: Actively secure OT extension with optimal overhead. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 724–741. Springer, Heidelberg (Aug 2015). [https://doi.org/10.1007/978-3-662-47989-6\\_35](https://doi.org/10.1007/978-3-662-47989-6_35)
  49. Keller, M., Orsini, E., Scholl, P.: MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 830–842. ACM Press (Oct 2016). <https://doi.org/10.1145/2976749.2978357>
  50. Kitaev, A.Y.: Quantum measurements and the abelian stabilizer problem. *Electron. Colloquium Comput. Complex.* **TR96-003** (1996), <https://eccc.weizmann.ac.il/eccc-reports/1996/TR96-003/index.html>
  51. Lai, Y.F., Galbraith, S.D., Delpéch de Saint Guilhem, C.: Compact, efficient and UC-secure isogeny-based oblivious transfer. In: Canteaut, A., Standaert, F.X.

- (eds.) EUROCRYPT 2021, Part I. LNCS, vol. 12696, pp. 213–241. Springer, Heidelberg (Oct 2021). [https://doi.org/10.1007/978-3-030-77870-5\\_8](https://doi.org/10.1007/978-3-030-77870-5_8)
52. Maino, L., Martindale, C., Panny, L., Pope, G., Wesolowski, B.: A direct key recovery attack on SIDH. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part V. LNCS, vol. 14008, pp. 448–471. Springer, Heidelberg (Apr 2023). [https://doi.org/10.1007/978-3-031-30589-4\\_16](https://doi.org/10.1007/978-3-031-30589-4_16)
  53. Montgomery, H., Zhandry, M.: Full quantum equivalence of group action DLog and CDH, and more. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022, Part I. LNCS, vol. 13791, pp. 3–32. Springer, Heidelberg (Dec 2022). [https://doi.org/10.1007/978-3-031-22963-3\\_1](https://doi.org/10.1007/978-3-031-22963-3_1)
  54. Moriya, T., Onuki, H., Takagi, T.: SiGamal: A supersingular isogeny-based PKE and its application to a PRF. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part II. LNCS, vol. 12492, pp. 551–580. Springer, Heidelberg (Dec 2020). [https://doi.org/10.1007/978-3-030-64834-3\\_19](https://doi.org/10.1007/978-3-030-64834-3_19)
  55. Naor, M., Pinkas, B.: Oblivious transfer and polynomial evaluation. In: Vitter, J.S., Larmore, L.L., Leighton, F.T. (eds.) Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA. pp. 245–254. ACM (1999). <https://doi.org/10.1145/301250.301312>
  56. Page, A., Robert, D.: Introducing clapoti(s): Evaluating the isogeny class group action in polynomial time (2023), <https://eprint.iacr.org/2023/1766>
  57. Peikert, C.: He gives C-sieves on the CSIDH. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part II. LNCS, vol. 12106, pp. 463–492. Springer, Heidelberg (May 2020). [https://doi.org/10.1007/978-3-030-45724-2\\_16](https://doi.org/10.1007/978-3-030-45724-2_16)
  58. Rindal, P., Schoppmann, P.: VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part II. LNCS, vol. 12697, pp. 901–930. Springer, Heidelberg (Oct 2021). [https://doi.org/10.1007/978-3-030-77886-6\\_31](https://doi.org/10.1007/978-3-030-77886-6_31)
  59. Robert, D.: Breaking SIDH in polynomial time. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part V. LNCS, vol. 14008, pp. 472–503. Springer, Heidelberg (Apr 2023). [https://doi.org/10.1007/978-3-031-30589-4\\_17](https://doi.org/10.1007/978-3-031-30589-4_17)
  60. Rostovtsev, A., Stolbunov, A.: Public-Key Cryptosystem Based On Isogenies. Cryptology ePrint Archive, Report 2006/145 (2006), <https://eprint.iacr.org/2006/145>
  61. Wesolowski, B.: Orientations and the supersingular endomorphism ring problem. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part III. LNCS, vol. 13277, pp. 345–371. Springer, Heidelberg (May / Jun 2022). [https://doi.org/10.1007/978-3-031-07082-2\\_13](https://doi.org/10.1007/978-3-031-07082-2_13)
  62. Zhang, F., Safavi-Naini, R., Susilo, W.: An efficient signature scheme from bilinear pairings and its applications. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 277–290. Springer, Heidelberg (Mar 2004). [https://doi.org/10.1007/978-3-540-24632-9\\_20](https://doi.org/10.1007/978-3-540-24632-9_20)

## A Security reductions

We prove the security reductions claimed in equation (6). We write  $A \leq B$  ( $A$  reduces to  $B$ ) to indicate that an algorithm against problem  $B$  can be used as a subroutine against problem  $A$ . We further write  $A \equiv B$  ( $A$  is equivalent to  $B$ ), if  $A \leq B$  and  $B \leq A$ .

As an auxiliary tool, we further restate the following problem, introduced in [34] and underlying the OT protocol in [51]. Note that in these sources, the problem is referred to as the *Inverse-CDH* problem.

*Problem 6 (Negative-CDH).* Given  $(E, [a]E)$  where  $a \in \mathbb{Z}_N$  and  $E \in \mathcal{E}$  and  $E \neq E_0$ , compute  $[-a]E$ .

It is immediate that the problems mentioned in Section 4.1, as well as Problem 6, reduce to GAIP (Def. 1), since knowledge of the secret isogenies allows to act with (multiples of or powers of) them. In [34], it is also proven that Negative-CDH  $\equiv$  Scalar-CDH. Furthermore, CDH  $\equiv$  Scalar-CDH under the assumption that the acting group has odd class number, which is always the case, when  $p \equiv 3 \pmod{4}$  [22], as is the case for us.

For the rest of this section, let us introduce the oracles  $\mathcal{O}^{\text{CDH}}$ ,  $\mathcal{O}^{\text{Scal}}$ ,  $\mathcal{O}^{\text{Exp}}$  and  $\mathcal{O}^{\text{Mult}}$  for the CDH, Scalar-CDH, Exp-CDH and Mult-CDH problems, respectively. We write oracle queries as e.g.  $[ab]E \leftarrow \mathcal{O}^{\text{Mult}}(E, [a]E, [b]E)$ . We state the obvious reductions first:

- Scalar-CDH  $\leq$  Mult-CDH: On input  $(c, E, [a]E)$ , query  $[ca]E \leftarrow \mathcal{O}^{\text{Mult}}(E, [a]E, [c]E)$  and return  $[ca]E$ .
- Exp-CDH  $\leq$  Mult-CDH: On input  $(E, [a]E, e)$ , write  $e$  in binary form and use a square-and-multiply type approach with queries of the form  $[a^{i+j}]E \leftarrow \mathcal{O}^{\text{Mult}}(E, [a^i]E, [a^j]E)$ .<sup>8</sup> This solves Exp-CDH with at most  $2 \log e$  queries to  $\mathcal{O}^{\text{Exp}}$ .

We continue by stating an algorithm for Negative-CDH  $\leq$  Exp-CDH. The idea behind the algorithm is to search for a number  $b = a + c$  and an exponent  $e$ , such that  $b^e \equiv -1 \pmod{N}$ . We do this, by starting from Euler's theorem and incrementally looking for a  $b$  satisfying this property.

1. On input  $(E, [a]E)$ , compute  $[1]E$  and  $[-1]E$ , and define  $b = a$ .
2. Set  $e = \phi(N)$ , where  $\phi$  is the Euler totient function.
3. Query  $F \leftarrow \mathcal{O}^{\text{Exp}}(E, [b]E, e/2^i)$ , by incrementing  $i$  from 0 upwards, until  $F \neq [1]E$ .
4. If  $F \neq [-1]E$ , repeat from step 2. with  $b \leftarrow b + 1$ , by constructing  $[b]E = [1][a]E$ . Keep track of  $c = b - a$ .
5. If  $F = [-1]E$ , we have found an  $e$ , such that  $[b^e]E = [-1]E$ , thus we can query  $[-b]E \leftarrow \mathcal{O}^{\text{Exp}}(E, [b]E, e + 1)$  and recover  $[-a]E = [c][-b]E$ .

<sup>8</sup> Even if trivial queries of the type  $\mathcal{O}^{\text{Mult}}(E, [a]E, [a]E)$  are prohibited, an adversary can sample  $k \leftarrow \mathbb{Z}_N$  and query the oracle with  $[a^2 - k^2]E \leftarrow \mathcal{O}^{\text{Mult}}(E, [a - k]E, [a + k]E)$ , from which it can easily recover  $[a^2]E = [k^2][a^2 - k^2]E$ , while the queried curves will look completely random to the oracle.

It is clear that the complexity of this algorithm depends on the number of divisors of  $N$ . Heuristically, we assume steps 2 through 4 to be repeated  $O(2^r)$  times, where  $r$  is the number of divisors of  $N$ .<sup>9</sup>

Finally, we show that  $\text{Mult-CDH} \leq \text{Exp-CDH}$  for odd  $N$ , implying equivalence in our setting. To this end, we give the adversary against  $\text{Mult-CDH}$  access to  $\mathcal{O}^{\text{Exp}}$  as well as to  $\mathcal{O}^{\text{CDH}}$  and  $\mathcal{O}^{\text{Scal}}$ . Note that the latter two can be implemented through  $\mathcal{O}^{\text{Exp}}$ .

1. On input  $(E, [a]E, [b]E)$ , compute  $[a + b]E \leftarrow \mathcal{O}^{\text{CDH}}(E, [a]E, [b]E)$ .
2.  $[a^2 + 2ab + b^2]E \leftarrow \mathcal{O}^{\text{Exp}}(E, [a + b]E, 2)$ .
3.  $[a^2]E \leftarrow \mathcal{O}^{\text{Exp}}(E, [a]E, 2)$  and  $[-a^2]E \leftarrow \mathcal{O}^{\text{Scal}}(-1, E, [a^2]E)$ ,  
 $[b^2]E \leftarrow \mathcal{O}^{\text{Exp}}(E, [b]E, 2)$  and  $[-b^2]E \leftarrow \mathcal{O}^{\text{Scal}}(-1, E, [b^2]E)$
4.  $[2ab + b^2]E \leftarrow \mathcal{O}^{\text{CDH}}(E, [a^2 + 2ab + b^2]E, [-a^2]E)$ ,  
 $[2ab]E \leftarrow \mathcal{O}^{\text{CDH}}(E, [2ab + b^2]E, [-b^2]E)$ .
5. Finally return  $[ab]E \leftarrow \mathcal{O}^{\text{Scal}}(E, [2ab]E, 1/2 \bmod N)$ .

We end up with the picture

$$\text{Neg-CDH} \equiv \text{Scalar-CDH} \equiv \text{CDH} \leq \text{Exp-CDH} \equiv \text{Mult-CDH} \leq \text{GAIP},$$

which holds for odd  $N$ .

## B Algorithms for exponentiation

### B.1 Exponentiation with trusted setup

The same idea as multiplication with Beaver triples can be used in order to prove elements in  $\mathcal{L}^{\text{Exp}}$ , e.g. compute powers of a secret  $a \in \mathbb{Z}_N$ , such as  $[a^e]E_0$ , for some  $e \in \mathbb{N}$ . We can use the same trick to compute

$$[a^2]E_0 = [(\alpha + x)^2]E_0 = [\alpha^2 + 2\alpha x + x^2]E_0,$$

where  $x \leftarrow \mathbb{Z}_N$  and  $\alpha = a - x$ . By allowing access to tuples of the type

$$B = (\text{id}; x, M_x, M_{x^2}) = (\text{id}; x, [x]E_0, [x^2]E_0) \in \mathbb{Z}_N \times \mathcal{E}^2,$$

where  $\text{View}_P(B) = B$  and  $\text{View}_V(B) = (\text{id}, M_x, M_{x^2})$ , we can define the squaring protocol of Figure 11. Similarly, precomputed tuples of the type

$$B = (\text{id}; x, M_x, M_{x^2}, M_{x^3}) = (\text{id}; x, [x]E_0, [x^2]E_0, [x^3]E_0) \in \mathbb{Z}_N \times \mathcal{E}^3$$

with  $\text{View}_P(B) = B$  and  $\text{View}_V(B) = (\text{id}, M_x, M_{x^2}, M_{x^3})$  allow cubing in the exponent via

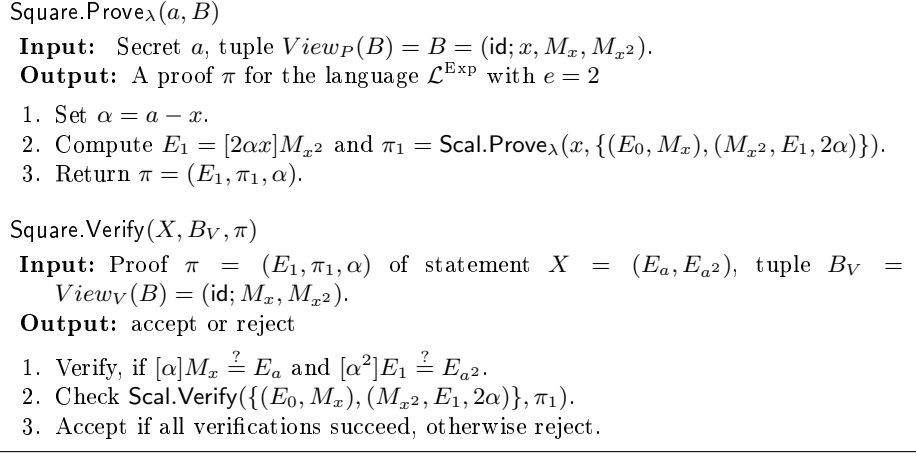
$$[a^3]E_0 = [(\alpha + x)^3]E_0 = [\alpha^3 + 3\alpha x^2 + 3\alpha^2 x + x^3]E_0,$$

which we describe in Figure 12. Security of these protocols can be proven in exactly the same way as in Theorem 2 and is therefore omitted.

For any exponent  $e > 3$ , a square-and-multiply type approach to compute  $[a^e]E_0$  can use `Square.Prove` and `Mult.Prove` consecutively.<sup>10</sup>

<sup>9</sup> Since we are assuming  $N$  to not be smooth, this reduction is in general quite efficient.

<sup>10</sup> We note that in some specific cases, tripling might be quicker.



**Fig. 11.** Non-interactive zero-knowledge proof and verification squaring in the exponent.

*Cost.* We can see that squaring costs  $1 + 2\lambda$  group actions for the prover and  $2 + 2\lambda$  for the verifier, while tripling costs  $2 + 4\lambda$  for the prover and  $3 + 4\lambda$  for the verifier, thus the same as multiplication. For a generic  $e$ , we will need at most  $\lfloor \log_2 e \rfloor$  squarings and multiplications, resulting in the total cost of at most  $\lfloor \log_2 e \rfloor (3 + 6\lambda)$  for the prover and  $\lfloor \log_2 e \rfloor (5 + 6\lambda)$  for the verifier.

## B.2 Exponentiation-in-the-Head

In this section, we outline the full protocol for exponentiation with a trusted setup, as introduced in Section 4.4.2. The protocol uses the in-the-Head protocol  $\text{ExplTH}_n^e(\alpha, B_P)$  from Figure 6 as a subroutine. We again assume that the prover can give access to tuples  $B_i = (\text{id}, (x_i^{(k)})^{k \in [e]})$  to the verifier.

We state security below. We only sketch the proof of this theorem as it works analogous to the proof of Theorem 3.

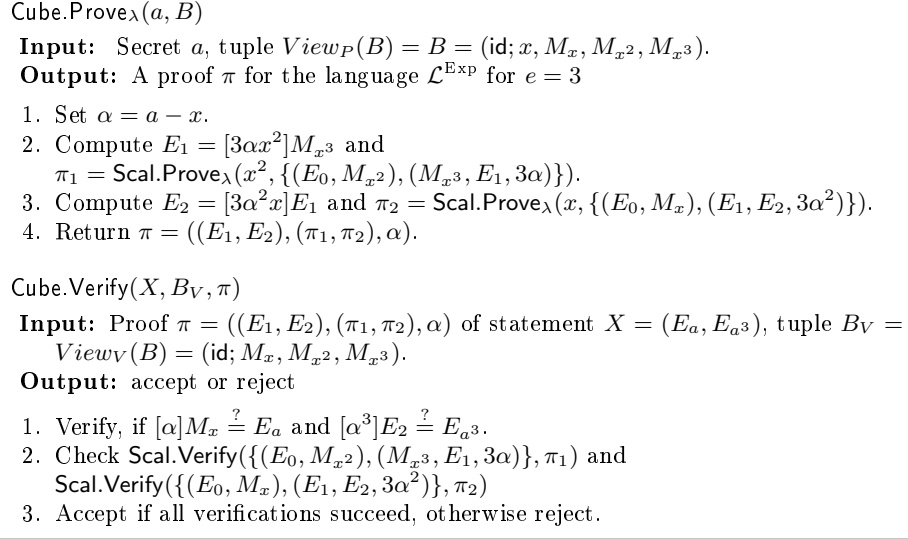
**Theorem 6.** *Assuming ideal commitments, the protocol in Figure 13 is an interactive and honest-verifier zero-knowledge proof of knowledge for the language  $\mathcal{L}^{\text{Exp}}$  with soundness error  $1/n$ .*

*Proof (Sketch).* **Completeness** follows from

$$E_n = \left[ \alpha^e + \sum_{k=1}^e \binom{e}{k} \alpha^{e-k} x_i^{(k)} \right] E_0 = [(\alpha + x)^e] E_0 = [a^e] E_0.$$

**Special soundness.** Analogous to the multiplicative case, from two accepting transcripts with  $c \neq c'$ , we can extract  $x$  from  $\{B_i\}_i \in [n]$  and extract  $a$ .

**Zero-knowledge.** The simulator  $\mathcal{S}$  samples  $\alpha \leftarrow \mathbb{Z}_N$  at random and sets  $M_x = [-\alpha]E_a$ . Then for a challenge  $c$ , it computes  $F_0 = M_x, F_1, \dots, F_{c-1}$  correctly



**Fig. 12.** Non-interactive zero-knowledge proof and verification for cubing in the exponent.

“forwards” and  $F_n = E_{a^e}, F_{n-1}, \dots, F_c$  correctly “backwards”, so that all the verification conditions succeed. Indistinguishability is guaranteed through the same arguments as in the proof of Theorem 3.

*Cost.* The prover computes  $n$  group actions, one for each party, per protocol repetition. The verifier verifies  $n - 1$  of those and also verifies whether  $[\alpha]E_x = E_a$ , also resulting in the total of  $n$ . Since  $\tau = \lceil \lambda / \log_2 n \rceil$ , this cost is minimal for  $n = 3$  and results in a cost of approximately  $1.89\lambda$  isogeny computations per party, which is faster than the multiplication approaches, and faster than proofs of DH-tuples. Unlike in the square-and-multiply approach presented in the previous section, this cost is independent of the exponent  $e$ . However, the downside of large  $e$  is that we need many precomputed tuples  $x_i^{(k)}$ . Yet, square-and-multiply type approaches would also be possible using the MPCitH protocols from this section, where the cost would increase, but the number of precomputed elements would scale with  $\log_2 e$ .

### B.3 Removing the trusted setup from Exponentiation-in-the-Head.

In Figure 14, we apply the cut-and-choose technique to the proof of exponentiation in  $\mathcal{L}^{\text{Exp}}$ .

**Theorem 7.** *Assuming ideal commitments, the protocol in Figure 14 is an interactive and honest-verifier zero-knowledge proof of knowledge for the language  $\mathcal{L}^{\text{Exp}}$  with soundness error  $\epsilon_{\text{ChC}}(m, n, \tau)$ .*

**Input:** Secret  $a$ , statement  $X = \{e, [a]E_0, [a^e]E_0\}$ , security parameter  $\lambda$ .  
**Output:** accept or reject whether  $X \in \mathcal{L}^{\text{Exp}}$ .

**Prover:**

1. Query  $\mathcal{T}$  for tuple  $B_P = (\text{id}; (x_i^{(k)})_{i \in [n]}^{k \in [e]})$ .
2. Execute  $(\mathcal{V}_1, \dots, \mathcal{V}_n) \leftarrow \text{ScalarITH}_n^e(\alpha, B_P)$ ,  
with  $\alpha = a - x$  and  $\beta = b - y$ .
3. For  $i = 1, \dots, n$ , sample  $\mu_i \leftarrow \{0, 1\}^\lambda$  and commit to  $C_i = \mathcal{C}(\mathcal{V}_i, \mu_i)$ .  
Send  $\{C_1, \dots, C_n\}$  and  $\alpha$  to the verifier.

**Verifier:**

4. Sample a challenge  $c \in [n]$  uniformly at random and send  $c$  to the prover.

**Prover:**

5. Send  $\{\mathcal{V}_i, \mu_i\}_{i \in [n] \setminus \{c\}}$ .

**Verifier:**

6. Check that all  $\mathcal{V}_i$ , for  $i \in [n] \setminus \{c\}$ , contain the same id.
7. Query  $\mathcal{T}$  with id for tuple  $B_V = (\text{id}, M_x)$ .
8. Check  $[\alpha]M_x \stackrel{?}{=} E_a$ .
9. Check the commitments  $C_i \stackrel{?}{=} \mathcal{C}(\mathcal{V}_i, \mu_i)$ , for  $i \in [n] \setminus \{c\}$ .
10. If  $c \neq 1$ , verify that  $F_0 \stackrel{?}{=} E_0$ , where  $F_0 \in \mathcal{V}_1$ .
11. If  $c \neq n$ , verify that  $F_n \stackrel{?}{=} E_a^e$  and  $F_n \stackrel{?}{=} \left[ \alpha^e + \sum_{k=1}^e \binom{e}{k} \alpha^{e-k} x_n^{(k)} \right] F_{n-1}$ ,  
where  $F_{n-1}, F_n \in \mathcal{V}_n$  and  $(x_n^{(k)})^{k \in [e]} \in B_n$ .
12. For  $i \in [n-1] \setminus \{c\}$ ,
  - (a) if  $i \neq n$ , verify that  $F_i = \left[ \sum_{k=1}^e \binom{e}{k} \alpha^{e-k} x_i^{(k)} \right] F_{i-1}$ , for  $F_{i-1}, F_i \in \mathcal{V}_i$  and  $(x_i^{(k)})^{k \in [e]} \in B_i$ .
  - (b) if  $c \neq i+1$ , verify that  $F_i \in \mathcal{V}_i$  is equal to  $F_{i-1} \in \mathcal{V}_{i-1}$ .
13. If all checks succeed, return accept, otherwise reject.

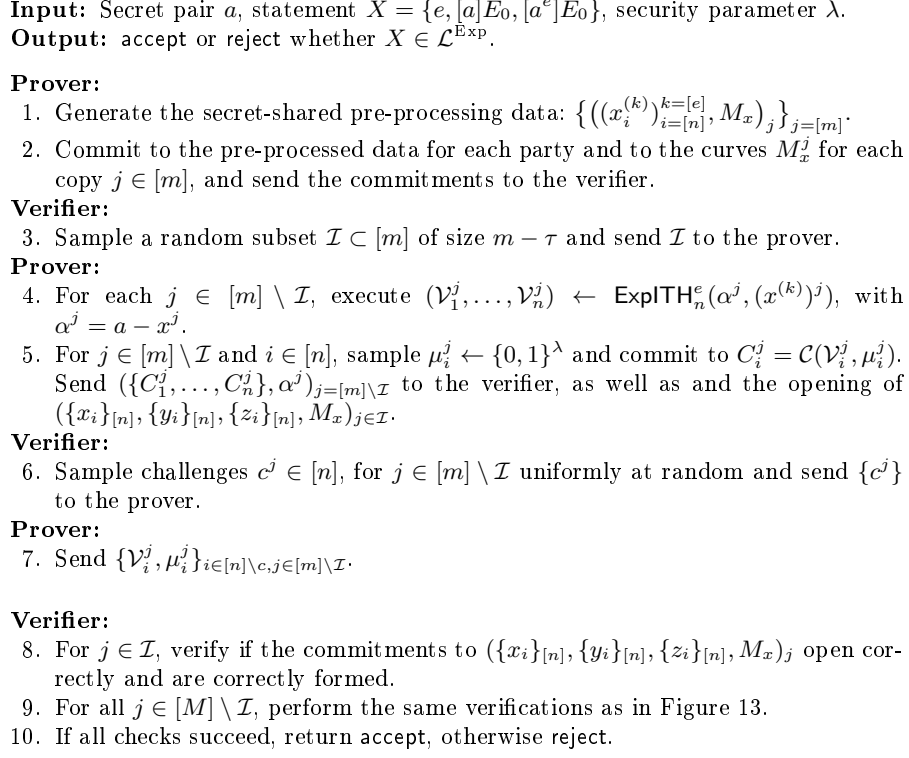
**Fig. 13.** Interactive ZK proof for  $\mathcal{L}^{\text{Exp}}$  using MPC-in-the-Head with trusted party.

*Proof. Correctness:* It is clear that correctly formed tuples pass the verification conditions for  $j \in \mathcal{I}$ . For each execution  $j \in [m] \setminus \mathcal{I}$ , correctness follows from the correctness of  $\text{ExpITH}$ .

**Special soundness:** By the same argument as for Theorem 6, for a given  $j$ , the extractor can reconstruct  $(x^{(1)})^j$  and extract  $a$  through  $\alpha^j$ . By further rewinding the prover back to the commitment of the pre-processing data, and obtaining a third accepting transcript with a different opening of  $\tau$  datasets, the extractor can ensure that the  $(x^{(1)})^j$  used above is a valid multiplication tuple.

**Zero-knowledge:** To output an indistinguishable transcript, the simulator  $\mathcal{S}$  first samples  $\mathcal{I}$  at random and then generates honest secret-sharings  $(\{x_i\}_{[n]}, \{y_i\}_{[n]}, \{z_i\}_{[n]}, M_x)_j$  for  $j \in \mathcal{I}$ . For the remaining  $j \in [m] \setminus \mathcal{I}$ ,  $\mathcal{S}$  does as for Theorem 3 by sampling  $\alpha^j$  at random and setting  $M_x^j = [-\alpha^j]E_a$ . It then falsifies the round-robin computation of  $F_n^j$  in the same way: by sampling the challenge  $c^j$  at random, and computing  $F_{c-1}^j$  “forwards” from  $E_0$  and  $F_c^j$





**Fig. 14.** Interactive ZK proof for  $\mathcal{L}_1^{\text{Mult}}$  using MPCitH without trusted party.

“backwards” from  $E_{a^e}$ . Since the  $(x_c^{(k)})^j$  remain hidden from the Verifier, and since there is no commitment as in the protocol with trusted setup, no other simulation is necessary.

*Correctness:* Here,  $F_n^j = E_{a^e}$  by construction and the views will be consistent due to the “forward” and “backward” computations.

*Indistinguishability:* The public values  $\alpha^j$  are sampled at random, which means they are distributed identically to the protocol, assuming that the Verifier has no knowledge of  $(x_c^{(k)})^j$ . Similarly, the  $M_x^j$  are distributed identically to an honest tuple. The idealised commitments perfectly hide  $(x_c^{(k)})^j$  from the Verifier, so the perfect secrecy of the additive secret-sharing scheme implies the perfect zero-knowledge of the protocol.  $\square$