Sashimi: Cutting up CSI-FiSh secret keys to produce an actively secure distributed signing protocol

Daniele Cozzo[1[0000−0001−5289−3769]] and Nigel P. Smart[1,2[0000−0003−3567−3304]]

[1] imec-COSIC, KU Leuven, Leuven, Belgium.
[2] University of Bristol, Bristol, UK.
daniele.cozzo@kuleuven.be, nigel.smart@kuleuven.be

Abstract. We present the first actively secure variant of a distributed signature scheme based on isogenies. The protocol produces signatures from the recent CSI-FiSh signature scheme. Our scheme works for any access structure, as we use a replicated secret sharing scheme to define the underlying secret sharing; as such it is only practical when the number of maximally unqualified sets is relatively small. This, however, includes the important case of full threshold, and (n, t)-threshold schemes when n is small.

1 Introduction

Threshold signature schemes have recently received more and more attention due to applications in blockchain and other scenarios where high value signatures are produced. Apart from early work on threshold RSA signatures [Sho00, DK01] and DSA/EC-DSA signatures [GJKR96, MR01], we have seen renewed interest in methods to produce EC-DSA signatures [CDK+18, DKLs18, GGN16, GG18, Lin17, LN18, LNR18], and interest in threshold schemes from standards bodies such as NIST [BDV19].

In the post-quantum world there has obviously been less work on this problem. In [CS19] Cozzo and Smart discuss the possibilities for threshold-izing the Round 2 candidate signature schemes in the NIST post-quantum 'competition'. The authors conclude that virtually all proposed signature schemes, with the possible exception of those based on the MQ-like problems, are hard to efficiently turn into threshold variants. However, the NIST candidates do not include any submission based on isogenies; mainly because isogeny based signature schemes did not become efficient until *after* the NIST 'competition' started.

Isogeny based cryptography goes back to the work of Couveignes, Rostovtsev and Stolbunov [Cou06, RS06]. The first isogeny based signature scheme was created by Stolbunov in his thesis [Sto12]. The basic construction was a Fiat-Shamir transform applied to a standard three-round isogeny-based identification scheme. The problem with Stolbunov's scheme is that one required an efficient method to sample in the class group, and that each class group member should have an efficiently computable unique representation.

To solve these problems De Feo and Galbraith used the Fiat-Shamir with aborts method to produce a new signature scheme, based on Stolbunov's, called SeaSign [DG19]. The SeaSign scheme was further improved by Decru et al [DPV19]. However, the algorithm still required two minutes to sign a single message.

Recently, Beullens at el [BKV19] returned to Stolbunov's original method and by calculating the ideal class group of an imaginary quadratic number field with large discriminant were able to instantiate the signature scheme efficiently. This instantiation of Stolbunov's scheme, called CSI-FiSh, requires only 390ms to sign or verify a message, and has signature sizes of only 263 bytes. Thus with CSI-FiSh isogeny based signatures are truly practical.

In [FM19] De Feo and Meyer consider the case of making CSI-FiSh into a threshold scheme, by distributing the secret key using the Shamir secret sharing scheme. Their resulting protocol is efficient, but only *passively* secure. The main trick that De Feo and Meyer use is to overcome the difficulty that isogenies can be composed, but do not form a group. As a result, performing the calculation of the signature will be more challenging than in the classic setting of distributed signatures based on discrete logarithms. Distributed signing protocols typically have each signer producing a partial signature which is then combined non-interactively into the final signature. Instead, in both the protocol of De Feo and Meyer and our protocol, the signature is produced more in the fashion of a ring signature, with each signer needing to accept and receive a message. A major simplification in our presentation is that we use a Replicated Secret Sharing Scheme. This means that, for a given qualified set, we can treat the resulting sharing as a full threshold sharing.

Just as CSI-FiSh follows the Fiat-Shamir paradigm in defining a signature scheme from isogenies, in much the same way as Schnorr signatures are created from discrete logarithms, we can follow the same paradigm in creating an actively secure threshold variant as is done in the standard case of actively secure distributed Schnorr signatures. Each signer, in the qualified set being used to sign, attaches a zero-knowledge proof to their partial signatures. This ensures the signer has followed the protocol, and importantly for our simulation proof it allows the simulator to extract the underlying secret witness. A similar strategy is used for simulating the key generation.

As just indicated, we prove our protocol secure in a simulation paradigm, but not in a the Universal Composability setting. This is because our protocol makes extensive use of Σ-protocols and the simulator needs to rewind the adversary in order to perform knowledge extraction from the special soundness of the underlying Σ-protocols. Thus our protocol should only be considered 'stand-alone' secure.

We estimate that our protocol will require just under five minutes to execute for the important cases of two party signing, or threshold signing with $(n, t) = (3, 1)$. This cost is mainly due to the zero-knowledge proofs needed to provide active security for our signing protocol.

Improvements to our work could be performed in a number of directions. On a theoretical front a fully UC protocol and proof would be interesting. A method to produce active security, in the standalone setting, without recourse to our zero-knowledge proofs would obviously have a big affect on performance. Extending our method to create an actively secure variant of the Shamir based protocol of De Feo and Meyer should be relatively easy. A change to our zero-knowledge proof technique would be of great interest, although this seems particularly hard, as any improvement to that would likely result in a major performance improvement in the basic CSI-FiSh signature scheme as well.

2 Preliminaries

2.1 Notation

We assume that all involved parties are probabilistic polynomial time Turing machines. Given a positive integer n, we denote by $[n]$ the set $\{1, \ldots, n\}$. We let $x \leftarrow X$ denote the uniformly random assignment to the variable x from the set X, assuming a uniform distribution over X. We also write $x \leftarrow y$ as shorthand for $x \leftarrow \{y\}$. If $\mathcal{D}$ is a probability distribution over a set X, then we let $x \leftarrow \mathcal{D}$ denote sampling from X with respect to the distribution $\mathcal{D}$. If A is a (probabilistic) algorithm then we denote by $a \leftarrow A$ the assignment of the output of A where the probability distribution is over the random tape of A.

2.2 Replicated Secret Sharing

Let $\mathcal{P} = \{P_i\}_{i=1,\ldots,n}$ be the set of parties and let $\Gamma \subset 2^{\mathcal{P}}$ be a monotone family for the relation of inclusion, that is if $Q \in \Gamma$ and $Q \subset Q'$ then $Q' \in \Gamma$. Similarly, let $\Delta \subset 2^{\mathcal{P}}$ be a monotone family with respect to the relation of subsets, that is if $U \in \Delta$ and $U' \subset U$ then $U' \in \Delta$. The pair (Δ, Γ) is called a *monotone access structure* if it holds that $\Delta \cap \Gamma = \emptyset$. We will only consider access structures where Δ and Γ are complementary to each other. The sets inside Γ are called *qualified sets* while the one in Δ are called *unqualified sets*. We denote by Γ^- the family of minimally qualified sets in Γ with respect to the inclusion relation, that is

$$\Gamma^- = \left\{ Q \in \Gamma : Q' \in \Gamma, Q' \subset Q \Rightarrow Q' = Q \right\}.$$

Similarly, we define the family of maximally unqualified sets Δ^+ as

$$\Delta^+ = \left\{ U \in \Gamma : U' \in \Delta, U \subset U' \Rightarrow U' = U \right\}.$$

Let Γ be a general monotone access structure and let R be a ring. The replicated scheme for Γ is defined as in Figure 1. To define the replicated scheme we first define a set $\mathcal{B} = \left\{ B \in 2^{\mathcal{P}} : \mathcal{P} \setminus B \in \Delta^+ \right\}$, then to share a secret $s \in R$ the dealer first additively shares $s = s_{B_1} + \ldots + s_{B_t}$ for $B_i \in \mathcal{B}$. To open a secret is straightforward. For each qualified set Q we define a mapping $\Psi_Q : \mathcal{B} \longrightarrow \mathcal{P}$ which allows the parties in Q to *uniquely* treat their shares as a full threshold sharing of the secret. In particular for each Q we require

$$s = \sum_{P_i \in Q} \left(\sum_{\Psi_Q(B) = P_i} s_B \right),$$

i.e. Ψ_Q partitions the shares s_B for $B \in \mathcal{B}$ between the parties in Q. Such replicated secret sharing schemes are clearly linear, and we will denote sharing an element by this secret sharing scheme by $\langle s \rangle$.

For a set of adversarys $\mathcal{A} \in \Delta$ we can divide the sets $B \in \mathcal{B}$, and hence shares s_B, into three disjoint sets $\mathcal{B}_A$, $\mathcal{B}_M$ and $\mathcal{B}_H$; $\mathcal{B} = \mathcal{B}_A \cup \mathcal{B}_M \cup \mathcal{B}_H$. The sets B in $\mathcal{B}_A$ correspond to shares s_B that are held *only* by the adversary, those in $\mathcal{B}_H$ are those held *only* by honest parties, whist those in $\mathcal{B}_M$ are held by *a mixture* of honest and adversarial parties. For all secret sharing schemes we have $\mathcal{B}_H \neq \emptyset$, otherwise we would have $\mathcal{A} = \mathcal{P}$. In the case of full-threshold sharing we always have $\mathcal{B}_M = \emptyset$.

2.3 Commitment Schemes

Our protocols require access to a commitment functionality $\mathcal{F}_{\mathsf{Commit}}$. The commitment functionality is a standard functionality allowing one party to first commit, and then decommit, to a value towards another set of parties. We assume that the opened commitment is only available to the receiving parties (i.e. it is sent over a secure channel). The functionality is given in Figure 2, and it is known to be easily implemented in the random oracle model.

Replicated Secret Sharing over the ring R

Input: For party to share a secret input $s \in R$, it performs
- Sample $s_B \leftarrow R$ for $B \in \mathcal{B}$ subject to $\sum_{B \in \mathcal{B}} s_B = s$
- For each $B \in \mathcal{B}$ and each $P_j \in B$ give s_B to party P_j.

Open: For a qualified set of parties Q to open a secret s
- For each $B \in \mathcal{B}$ if $P_j \in Q$ and $P_j \notin B$ then all parties $P_i \in B$ send s_B to party P_j
- Each party computes $s = \sum_{B \in \mathcal{B}} s_B$.

ToFullThreshold: For a qualified set of parties Q
- For $P_i \in Q$ define $z_{P_i} \leftarrow \sum_{\Psi_Q(B)=P_i} s_B$.

Figure 1. Replicated Secret Sharing over the ring R

The Functionality $\mathcal{F}_{\mathsf{Commit}}$

Init: On input of (Init, P_i, B) from all players, this initializes a commitment functionality from player P_i to the players in B. We write this as $\mathcal{F}^{i,B}_{\mathsf{Commit}}$, if B is a singleton set $B = \{j\}$ then we write $\mathcal{F}^{i,j}_{\mathsf{Commit}}$, and if $B = \mathcal{P} \setminus \{i\}$ then we write $\mathcal{F}^{P_i}_{\mathsf{Commit}}$.

Commit: On input of $(\mathsf{Commit}, \mathsf{id}, \mathsf{data})$ from player P_i and $(\mathsf{Commit}, \mathsf{id}, \perp)$ from all players in B the functionality stores $(\mathsf{id}, \perp)$.

Open: On input of $(\mathsf{Commit}, \mathsf{id})$ from all players in $B \cup \{i\}$ the functionality retrieves the entry $(\mathsf{id}, \mathsf{data})$ and returns data to all parties in B.

Figure 2. The Functionality $\mathcal{F}_{\mathsf{Commit}}$

2.4 PRSSs

In our protocols we utilize the fact that, after a key distribution phase, parties can generate non-interactively sharings in a replicated scheme; namely we can define a so-called PRSS. In particular, we require the parties to engage in a pre-processing phase in which they share keys for a Pseudo-Random Function (PRF) in order to generate Pseudo-Random Secret Sharings (PRSSs) for the replicated scheme $\langle v \rangle$. In particular, we make black-box use of the functionality given in Figure 3. PRSSs for arbitrary access structures can involve a set-up phase requiring the agreement of exponentially-many keys in general. The general protocol is given in [CDI05]. To set up the PRSS in the case of our replicated scheme we use the method described in Figure 4, where $F_k(\cdot)$ is a PRF with codomain equal to R.

The Functionality $\mathcal{F}_{\mathsf{Rand}}$

Init: The functionality accepts Init or abort from all parties and the adversary. If any party inputs abort, the functionality sends the message abort to all parties.

PRSS: On input $\mathsf{PRSS}(\mathsf{cnt})$ from all parties, if the counter value is the same for all parties and has not been used before, the functionality samples a set $\{r_B\}_{B \in \mathcal{B}} \leftarrow R$ and for each $B \in \mathcal{B}$ sends r_B to all $i \in B$.

Figure 3. The Functionality $\mathcal{F}_{\mathsf{Rand}}$

Theorem 2.1. *Assuming F is a pseudo-random function, the protocol Π_{Rand} securely realises $\mathcal{F}_{\mathsf{Rand}}$ in the $\mathcal{F}_{\mathsf{Commit}}$-hybrid model.*

Protocol Π_{Rand}

Init: The parties initialise by doing the following:

1. For each $i \in \mathcal{P}$, for each $j \in \mathcal{P} \setminus \{i\}$, party i samples $\kappa_{i,j} \leftarrow \{0,1\}^\lambda$.
2. Each pair of parties initialises an instance of $\mathcal{F}_{\mathsf{Commit}}$; call it $\mathcal{F}^{i,j}_{\mathsf{Commit}}$.
3. For each $i \in \mathcal{P}$, for each $j \in \mathcal{P} \setminus \{i\}$, the parties call $\mathcal{F}^{i,j}_{\mathsf{Commit}}$ where i submits input $(\mathsf{Commit}, \mathsf{id}_{i,j}, \kappa_{i,j})$ and j gives $(\mathsf{Commit}, \mathsf{id}_{i,j}, \perp)$.
4. For each $i \in \mathcal{P}$, for each $j \in \mathcal{P} \setminus \{i\}$, the parties execute $\mathcal{F}^{i,j}_{\mathsf{Commit}}$ with input $(\mathsf{Open}, \mathsf{id}_{i,j})$ and abort if $\mathcal{F}_{\mathsf{Commit}}$ returns abort.
5. For each $B \in \mathcal{B}$, each party $i \in B$ samples $\kappa_{B,i} \leftarrow \{0,1\}^\lambda$.
6. Each set of parties $B \in \mathcal{B}$ initialises an instance of $\mathcal{F}_{\mathsf{Commit}}$; call it $\mathcal{F}^{i,B}_{\mathsf{Commit}}$.
7. For each $B \in \mathcal{B}$, for each $i \in B$, the parties call $\mathcal{F}^{i,B}_{\mathsf{Commit}}$ where i submits input $(\mathsf{Commit}, \mathsf{id}_{B,i}, \kappa_{B,i})$ and all $j \in B \setminus \{i\}$ give $(\mathsf{Commit}, \mathsf{id}_{B,i}, \perp)$.
8. For each $B \in \mathcal{B}$ and for each $i, j \in \mathcal{P}$ with $i, j \in B$, the parties i and j call $\mathcal{F}^{i,B}_{\mathsf{Commit}}$ with input $(\mathsf{Open}, \mathsf{id}_{B,i})$ and abort if $\mathcal{F}^{B}_{\mathsf{Commit}}$ returns abort.
9. Each party i sets $\kappa_B \leftarrow \oplus_{j \in B} \kappa_{B,j}$ for each B containing i.
10. Finally, each party i sets $\mathsf{cnt} \leftarrow 0$, and $\mathsf{cnt}_B \leftarrow 0$ for all $B \in \mathcal{B}$ where $i \in B$.

PRSS: For each $B \in \mathcal{B}$ containing i, party i computes

$$r_B \leftarrow F_{\kappa_B}(\mathsf{cnt}_B)$$

and increments cnt_B.

Figure 4. Protocol Π_{Rand}

Proof. The Init procedure is clearly secure assuming an secure commitment functionality. As there is no interaction after Init, the protocol is clearly secure if it is correct and passively secure. Correctness follows from basic algebra, and security follows from the fact that F is assumed to be a PRF and from the fact that there is at least one B not held by the adversary (by definition of the access structure). □

2.5 Elliptic Curves and Isogenies

In what follows E denotes an elliptic curve over a finite field $\mathbb{F}_p$ where p is a large prime. An elliptic curve is called supersingular if its number of rational points satisfies the equation $\#E(\mathbb{F}_p) \equiv 1 \pmod{p}$. An elliptic curve is called ordinary if this does not happen. An isogeny between two elliptic curves E and E' is a rational map $\varphi : E \to E'$ which is also a homomorphism with respect to the natural group structure of E and E'. An isomorphism between two ellliptic curves is an injective isogeny. The j-invariant of an elliptic curve is an algebraic invariant under isomorphism. As isogenies are group homomorphisms, any isogeny comes with a subgroup of E, which is its kernel. On the other hand, any subgroup $G \subset E\left(\mathbb{F}_{p^k}\right)$ yields a unique (up to automorphism) separable isogeny $\varphi : E \to E/G$ having G as kernel. It can be shown that the quotient E is an elliptic curve and its equation can be computed using standard formulae [V́71].

The set $\mathsf{End}(E)$ of all the isogenies of an elliptic curve E form a ring under the composition operator. The isogenies that can be written with coefficients in $\mathbb{F}_p$ forms a subring of $\mathsf{End}(E)$ and is denoted by $\mathsf{End}_{\mathbb{F}_p}(E)$. For supersingular elliptic curves this happens to be a proper subset. In particular, for supersingular elliptic curves the ring $\mathsf{End}(E)$ is an order of a quarternion algebra defined over $\mathbb{Q}$, while $\mathsf{End}_{\mathbb{F}_p}(E)$ is isomorphic to an order of the imaginary quadratic field $\mathbb{Q}(\sqrt{-p})$.

By abuse of notation we will identify $\mathsf{End}_{\mathbb{F}_p}(E)$ with the isomorphic order which we will denote by $\mathcal{O}$. The quotient of the fractional invertible ideals by the principal ideals in $\mathcal{O}$, denoted by $\mathsf{Cl}(\mathcal{O})$ of $\mathcal{O}$, is a group called class group of $\mathcal{O}$. There is a natural action of the class group on the class of elliptic curves defined over $\mathbb{F}_p$ with order $\mathcal{O}$. Given an ideal $\mathfrak{a} \subset \mathcal{O}$ one can define the subgroup $S_{\mathfrak{a}} = \cap_{\alpha \in \mathfrak{a}} \mathsf{Ker}(\alpha)$. As this is a subgroup of E one gets an isogenous elliptic curve $E/S_{\mathfrak{a}}$ defined up to $\mathbb{F}_p$-automorphism. We will denote the action of an element $\mathfrak{a} \subset \mathcal{O}$ on an elliptic curve E by $\mathfrak{a} \star E$. This action is free and transitive. This action is believed to be hard to invert, even for a quantum computer. Specifically, constructions based on the following problems are believed to be quantum secure:

Definition 2.1 (Group action inverse problem (GAIP) [DG19]). *Given two elliptic curves E and E' over the same finite field and with $\mathsf{End}(E) = \mathsf{End}(E') = \mathcal{O}$, find an ideal $\mathfrak{a} \subset \mathcal{O}$ such that $E' = \mathfrak{a} \star E$.*

There is a obvious decisional version of this problem, which we refer to as the decisional-GAIP, see [Sto12].

The CSI-FiSh signature scheme relies on the hardness of random instance of a multi-target version of GAIP, called MT-GAIP. In [DG19] it is shown that MT-GAIP reduces to GAIP when the class group structure is known.

Definition 2.2 (MT-GAIP). *Given k elliptic curves $E_1, \ldots, E_k$ over the same field, with $\mathsf{End}(E_1) = \cdots = \mathsf{End}(E_k) = \mathcal{O}$, find an ideal $\mathfrak{a} \subset \mathcal{O}$ such that $E_i = \mathfrak{a} \star E_j$ for some $i, j \in \{0, \ldots, k\}$ with $i \neq j$.*

2.6 Digital Signature Schemes

As is standard digital signature schemes are defined by

Definition 2.3. *A* digital signature scheme *is given by a tuple of probabilistic algorithms* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$*:*

- $\mathsf{KeyGen}\left(1^{\lambda}\right)$ *is a randomized algorithm that takes as input the security parameter and returns the public key* pk *and the private key* sk*.*
- $\mathsf{Sign}(\mathsf{sk}, \mu)$ *is a randomized signing algorithm that takes as inputs the private key and a message and returns a signature on the message.*
- $\mathsf{Verify}(\mathsf{pk}, (\sigma, \mu))$ *is a deterministic verification algorithm that takes as inputs the public key and a signature σ on a message μ and outputs a bit which is equal to one if and only if the signature on μ is valid.*

Correctness and security (EU-CMA) are defined in the usual way.

Definition 2.4. *Let $\mathcal{A}$ be an adversary that is given the public key* pk *and oracle access to the signing oracle* $\mathsf{Sign}_{\mathsf{sk}}$*. In its interaction with the oracle it can receive signatures on messages it adaptively chooses. Let $\mathcal{Q}$ be the set of of messages queried by $\mathcal{A}$. A digital signature scheme $\Pi =$* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *is said to be* existentially unforgeable *if there exists no such an adversary that can produce a signature on a message $m \notin \mathcal{Q}$, except with negligible probability in λ*

2.7 Distributed Signature Schemes

We assume the existence of secure point-to-point channels and synchronous channels, so parties receive data at the same time in a given round. For our adversarial model, we assume a malicious adversary that might deviate arbitrarily from the protocol. Given our access structure (Δ, Γ), the adversary can statically corrupt any non-qualified set. For a corrupted party, the adversary learns all the internal variables and controls both the input and the output ports of that party. Informally, our security requirement is that such an adversary will learn nothing about the underlying secret signing key, and that deviations from the protocol will result in an abort signal being sent to the honest parties.

Formally we define the ideal functionality given in Figure 5, and security is defined by requiring that for every adversary there is a simulator such that the adversary cannot tell if it is interacting in the real protocol, or if it is interacting with a simulator which has access to the ideal functionality. The ideal functionality is designed for a signature scheme in which the secret key is a vector of T elements in R, and the secret sharing of such keys is done via a replicated scheme. Note that, the ideal functionality allows the adversary to alter the sharing provided by the ideal functionality to a different secret key; however the ideal functionality then fixes this change to correspond to the public key initially generated. This cannot be detected by the adversary as the adversary does not see the public key until after it has made the change. This is consistent with how the adversary could attack an initial key distribution based on using a PRSS.

Distributed Signature Functionality: $\mathcal{F}_{\mathsf{DSign}}$

We let $\mathcal{A}$ denote the set of parties controlled by the adversary.

KeyGen: This proceeds as follows:

1. The functionality generates a public/private key pair; let the private key (say) be $\mathsf{sk} \in R^T$ and let the public key be pk.
2. The functionality shares sk via the replicated scheme as $\langle \mathsf{sk}_i \rangle'$, with $\mathsf{sk}_i = \sum_{B \in \mathcal{B}} s'_{i,B}$.
3. The values $s'_{i,B}$ are sent to player P_j for $P_j \in B$ for every $B \in \mathcal{B}_A \cup \mathcal{B}_M$.
4. The adversary enters shares $s_{i,B}$ for all share components in $\mathcal{B}_A$.
5. The functionality now defines $s_{i,B} = s'_{i,B}$ for all share components in $\mathcal{B}_M$.
6. The functionality completes the sharing so that it still shares sk_i by fixing the shares in $\mathcal{B}_H$ appropriately.
7. The value pk is output to the adversary.
8. The adversary returns with either abort or $\mathsf{deliver}$. If $\mathsf{deliver}$ the functionality returns pk to the honest parties, otherwise it aborts.

Sign: On input of a message m the functionality proceeds as follows:

1. The functionality adversary waits for an input from the adversary.
2. If the input is not abort then the functionality generates a signature σ on the message m.
3. The signature is returned to the adversary, and the functionality again waits for inpit. If the input is again not abort then the functionality returns σ to the honest players.

Figure 5. Distributed Signature Functionality: $\mathcal{F}_{\mathsf{DSign}}$

3 The CSI-FiSh signature scheme

In this section we recap on the basic CSI-FiSh signature scheme from [BKV19]. The scheme is defined in the isogeny graph of the public supersingular elliptic curve

$$E_0(\mathbb{F}_p) : y^2 = x^3 + x$$

where p is a prime of the form $p = 4 \cdot \ell_1 \cdot \cdots \cdot \ell_n - 1$, with ℓ_i being distinct small odd primes. For the set of primes $\ell_1, \ldots, \ell_{74}$, chosen in [CLM+18] for the CSIDH-512 parameter set, the authors of [BKV19] determine that the associated class group of the endomorphism ring is cyclic, generated by $\mathfrak{g}$, and has cardinality $N = \#\mathsf{Cl}(\mathcal{O})$ given by

$$\begin{aligned} N = 3 \times 37 \times 1407181 \times 51593604295295867744293584889 \\ \times 315994145046819958530082787455878322049 09. \end{aligned}$$

For any ideal $\mathfrak{a} \in \mathsf{Cl}(\mathcal{O})$ we can write $\mathfrak{a} = \mathfrak{g}^a$, where $a \in \mathbb{Z}/N\mathbb{Z}$, since the group is cyclic. Therefore we can identify uniquely the ideal $\mathfrak{a}$ with the integer a. To simplify notation we write, for an elliptic curve E' isogenous to E_0, $\mathfrak{a} \star E' = [a]E'$. With this notation we have $[a]([b]E) = [a+b]E$. For the elliptic curve E_0 it is also very easy to compute the quadratic twists. The quadratic twist E^t of the elliptic curve $E = [a]E_0$ is isomorphic over $\mathbb{F}_p$ to the elliptic curve $[-a]E_0$.

The basic identification scheme on which CSI-FiSh is built on starts with a public key being the action of a on the elliptic curve E_0, that is $E_1 := a \star E_0 = [a]E_0$. The prover starts by sampling a random element $b \in \mathbb{Z}/N\mathbb{Z}$, and sends the resulting commitment $[b]E_0$ to the verifier. This computation, according to [BKV19], takes around 40ms to compute per value of b. The verifier then samples a random challenge *bit* $c \in \{0, 1\}$ and returns it to the prover. The prover then responds with $r = b$ modulo N if $c = 0$ and with $r = b - a$ modulo N if $c = 1$. The verifier then checks that $[r]E_0 = E$ if $c = 0$ or $[r]E_1 = E$ if $c = 1$. This can then be turned into a signature scheme in the standard manner.

Having a binary challenge spaces gives an adversary a one in two chance of producing an invalid proof. One way to fix this is to enlarge the challenge space. This is done in [BKV19] as follows, which improves soundness, but increases the size of the public key. A positive integer S is chosen, with the secret key being a vector of dimension $S - 1$, say $(a_1, \ldots, a_{S-1})$ and with public key $(E_0, E_1 = [a_1]E_0, \ldots, E_{S-1} = [a_{S-1}]E_0)$. The prover now must prove that it knows a secret $s \in \mathbb{Z}/N\mathbb{Z}$ such that $E_j = [s]E_i$ for some pair of elliptic curves appearing in the public key list. The prover again chooses a random mask $b \in \mathbb{Z}/N\mathbb{Z}$ and commits to it via $E' = [b]E_0$. The verifier now samples the challenge c uniformly from the set $\{-S + 1, \ldots, S - 1\}$ and the prover responds with $r = b - a_c \pmod N$. Verification consists in checking that $[r]E_c = E'$, where we use the notation $E_{-c} = E_c^t$, for negative values. This variant of CSI-FiSh achieves soundness security $\frac{1}{2 \cdot S - 1}$. Thus to obtain $2^{-\mathsf{sec}}$ soundness security overall we need to repeat the basic protocol $t_{\mathsf{S}} = \mathsf{sec}/\log_2(2 \cdot S - 1)$ times, although one can reduce t_{S} a little bit by choosing a 'slow' hash function[3]

When combined with the Fiat–Shamir heuristic this gives the signature scheme presented in Figure 6, where $H : \{0, 1\}^* \longrightarrow [-S + 1, \ldots, S - 1]^{t_{\mathsf{S}}}$. This signature scheme is EU-CMA secure under the MT-GAIP assumption, when H is modelled as a random oracle.

[3] For highest computational efficiency [BKV19] selects, for $\mathsf{sec} = 128$, the values $S = 2^{15}$ and $t_{\mathsf{S}} = 7$, using a hash function which is 2^{16} times slower than SHA-3.

The CSI-FiSh Signature Algorithm

KeyGen: Key generation proceeds as follows:

1. For $i \in [1, \ldots, S-1]$ do
   (a) $a_i \leftarrow \mathbb{Z}/N\mathbb{Z}$.
   (b) $E_i \leftarrow [a_i]E_0$.
2. $\mathsf{sk} \leftarrow (a_1, \ldots, a_{S-1})$.
3. $\mathsf{pk} \leftarrow (E_0, E_1, \ldots, E_{S-1})$.

Sign(m, sk)**:** To sign a message m, the signer performs

1. For $i = 1, \ldots, t_{\mathsf{S}}$
   (a) $b_i \leftarrow \mathbb{Z}/N\mathbb{Z}$.
   (b) $E'_i \leftarrow [b_i]E_0$.
2. $(c_1, \ldots, c_{t_{\mathsf{S}}}) \leftarrow H(E'_1 \| \ldots \| E'_{t_{\mathsf{S}}} \| m)$.
3. For $i = 1, \ldots, t_{\mathsf{S}}$
   (a) $r_i \leftarrow b_i - \mathsf{sign}(c_i) \cdot a_{|c_i|} \pmod N$.
4. Output $\{(r_i, c_i)\}_{i=1}^{t_{\mathsf{S}}}$.

Verify$(\{(r_i, c_i)\}_{i=1}^{t_{\mathsf{S}}}, m, \mathsf{pk})$**:** To verify a signature $\{(r, c)\}_{i=1}^{t_{\mathsf{S}}}$ on a message m one performs

1. For $i = 1, \ldots, t_{\mathsf{S}}$ execute $E'_i \leftarrow [r_i]E_{c_i}$.
2. $(c'_1, \ldots, c'_{t_{\mathsf{S}}}) \leftarrow H(E'_1 \| \ldots \| E'_{t_{\mathsf{S}}} \| m)$.
3. If $((c_1, \ldots, c_{t_{\mathsf{S}}}) = (c'_1, \ldots, c'_{t_{\mathsf{S}}})$ then output one, else output zero.

Figure 6. The CSI-FiSh Signature Algorithm

3.1 Zero-Knowledge Proof

Our goal is to define a distributed signing protocol which is secure against malicious adversaries. To guarantee that the parties behave correctly, they are asked to commit to their secrets using the class group action and prove that what they are committing to is of the correct form. Clearly, to prove knowledge of a secret isogeny is sufficient to run an instance of the underlying basic CSI-FiSh identification scheme described above. However, we require to prove something a little more general, namely a witness s to the following relation, which we define for arbitrary j, but for which we only use when $j = 1$ and $j = 2$.

$$\mathbb{L}_j := \left\{ \left((E_1, E'_1, \ldots, E_j, E'_j),\ s \right) \ : \ \bigwedge_{i=1}^{j} \left(E'_i = [s]E_i \right) \right\}$$

In other words, the prover wants to prove in zero-knowledge that it knows a unique witness for j *simultaneous* instances of the GAIP. This can be done by using standard techniques of Σ-protocols. We present the underlying protocol in Figure 7. There are essentially two variants, one when $E_1 = \ldots = E_j = E_0$, and one when this condition does not hold. The call the first case the Special case, and the second case the the General case.

The following theorem shows that the basic interactive proof of knowledge has soundness error $1/2$ in the General case and $1/3$ in the Special case. Thus we need to repeat it sec (resp $\mathsf{sec}/\log_2 3$) times to achieve a security level of sec. In the random oracle this can be made non-interactive in the standard manner using a hash function G with codomain $\{0,1\}^{t_{\mathsf{ZK}}}$. Using a 'slow' hash function for G, as in the case of CSI-FiSh, which is 2^k times slower than a normal hash function we can reduce the number of repetitions to $t_{\mathsf{ZK}} = \mathsf{sec} - k$ (resp. $t_{\mathsf{ZK}} = (\mathsf{sec} - k)/\log_2 3$. When $k = 16$ and $\mathsf{sec} = 128$ as in the fastest CSI-FiSh parameters this gives us $t_{\mathsf{ZK}}^{\mathsf{General}} = 112$ for the General case and $t_{\mathsf{ZK}}^{\mathsf{Special}} = 70$ for the Special case. We denote the resulting non-interactive proof and verification algorithms by $\mathsf{ZK}.P$ and $\mathsf{ZK}.V$.

The prover $\mathsf{ZK}.P_i((E_1, E'_1, \ldots, E_j, E'_j), s)$ and verifier $\mathsf{ZK}.V_i(E_1, E'_1, \ldots, E_j, E'_j), \pi)$ functions for our zero-knowledge proof

$\mathsf{ZK}.P_1((E_1, E'_1, \ldots, E_j, E'_j))$**:** The first stage of the prover executes:

1. $b \leftarrow \mathbb{Z}/N\mathbb{Z}$.
2. For $i = 1, \ldots, j$ do $\hat{E}_i \leftarrow [b]E_j$.
3. Output $(\hat{E}_1, \ldots, \hat{E}_j)$.

$\mathsf{ZK}.V_1((E_1, E'_1, \hat{E}_1, \ldots, E_j, E'_j, \hat{E}_j), s)$**:** The first stage of the verifier is simply:

1. If $E_k \neq E_0$ for any $k \in \{1, \ldots, j\}$ then select $c \in \{0, 1\}$ and output it.
2. Else select $c \in \{-1, 0, 1\}$ and output it.

$\mathsf{ZK}.P_2((E_1, E'_1, \hat{E}_1, \ldots, E_j, E'_j, \hat{E}_j), c, s)$**:** The second stage of the prover executes:

1. $r \leftarrow b - c \cdot s \bmod N$, and outputs r.

$\mathsf{ZK}.V_2((E_1, E'_1, \hat{E}_1, \ldots, E_j, E'_j, \hat{E}_j), c, r)$**:** This algorithm gives the verifiers final calculation:

1. If $c = -1$ return $\bigwedge_{i=1}^{j} \left([r]E_i'^t = \hat{E}_i\right)$.
2. If $c = 0$ return $\bigwedge_{i=1}^{j} \left([r]E_i = \hat{E}_i\right)$.
3. If $c = 1$ return $\bigwedge_{i=1}^{j} \left([r]E'_i = \hat{E}_i\right)$.

Figure 7. The prover $\mathsf{ZK}.P_i((E_1, E'_1, \ldots, E_j, E'_j), s)$ and verifier $\mathsf{ZK}.V_i(E_1, E'_1, \ldots, E_j, E'_j), \pi)$ functions for our zero-knowledge proof

Theorem 3.1. *The interactive proof in Figure 7 is correct, has soundness error $\frac{1}{2}$ in the* General *case and soundness error $\frac{1}{3}$ in the* Special *case, and is computational zero-knowledge assuming decisional-GAIP.*

Proof. We first show correctness. We given the proof in the general case, as the special case follows similarly. Suppose that the prover behaves honestly; this means that it knows a secret s such that $E'_i = [s]E_i$ for all i. If $c = 0$ then verification consists in checking whether $[r]E_i = \hat{E}_i$ for all i. Since $[r]E_i = [b]E_i$ and this is equal to $\hat{E}_i$ for all i, the verifier accepts. If $c = 1$ then verification consists in checking whether $[r]E'_i = \hat{E}_i$ for all i. Since $[r]E'_i = [b - s]E'_i = [b - s]([s]E_i) = [b]E_i$ and this is equal to $\hat{E}_i$ the verifier accepts. This proves that the verifier always accepts an honestly generated proof.

To show soundness (again in the General case) we build an extractor using the standard technique. As E_i and E'_i are isogenous we can write $E'_i = [s]E_i$, for some unknown value $s \in \mathbb{Z}/N\mathbb{Z}$. After rewinding the prover, we obtain two accepting proofs of the form

$$\pi = \left((\hat{E}_1, \ldots, \hat{E}_j), c, r\right) \quad \text{and} \quad \pi' = \left((\hat{E}_1, \ldots, \hat{E}_j), c', r'\right)$$

where $c \neq c'$, and hence $r \neq r'$ (unless $s = 0$). Since the proofs accept we have, for all $i \in [1, \ldots, j]$,

$$[r]E_i = \hat{E}_i \quad \wedge \quad [r']E'_i = \hat{E}_i.$$

This implies that, for all i, we have

$$E'_i = [-r'][r']E'_i) = [-r'][r]E_i) = [r - r']E_i$$

which implies that $s = r - r'$ for all i. The extractor in the Special case is much the same.

To simulate the proof, one samples c at random from $\{0, 1\}$ for the General case and from $\{-1, 0, 1\}$ in the Special case. We also sample r at random from $\mathbb{Z}/N\mathbb{Z}$. One then sets $\hat{E}_i = [r]E_i$ if $c = 0$, $\hat{E}_i = E'_i$ if $c = 1$ and $\hat{E}_i = [r]E_i'^t$ if $c = -1$. In the case that the input to the proof

is from the language $\mathbb{L}_j$ then this simulation is perfect. If the input is not from the language $\mathbb{L}_j$ then the commitments also look like they come from a uniform distribution, because they are deterministic functions of the variable r which is uniform. However, the distribution of $\hat{E}_1, \ldots, \hat{E}_j$ is not correct. By the decisional version of the GAIP problem it is computationally hard for an adversary to distinguish the tuples $(\hat{E}_1, \ldots, \hat{E}_j)$ and $([b]E_1, \ldots, [b]E_j)$, and thus the simulation is computationally zero-knowledge.

4 A Threshold Implementation

In this section we show how to create an actively secure threshold implementation of the CSI-FiSh signature scheme for any access structure, where we hold the secret key using a replicated secret sharing scheme. Before doing so we present a useful sub-protocol for performing a full-threshold variant of the group action computation at the heart of isogeny based cryptography. See Figure 8 for the details; we defer the relevant proof of security till later. It uses the abstract (standard) commitment functionality $\mathcal{F}_{\mathsf{Commit}}$ given earlier. For later use we denote this sub-protocol by $\mathsf{GrpAction}(E_0, Q, [s])$, which for our fixed elliptic curve produces the group action $[s]Q$ in an actively secure manner.

Group Action Computation for a Full Threshold Secret Sharing

Input: The fixed elliptic curve E_0, a set of parties Q, a secret shared element $s \in \mathbb{Z}/N\mathbb{Z}$ held via a full threshold sharing, i.e. $P \in Q$ holds s_P such that $s = \sum_{P \in Q} s_P$.
Output: $[s]E_0$

1. Define an ordering the players in $Q = \{P_1, \ldots, P_t\}$.
2. Each party P_j initialises an instance of $\mathcal{F}_{\mathsf{Commit}}$; call it $\mathcal{F}_{\mathsf{Commit}}^{P_j}$.
3. For $j = 1, \ldots, t$
   - $E_{P_j} \leftarrow [s_{P_j}]E_0$.
   - $\pi^1_{P_j} \leftarrow \mathsf{ZK}.P((E_0, E_{P_j}), s_{P_j})$.
   - The parties call $\mathcal{F}_{\mathsf{Commit}}^{P_j}$ where P_j submits input $(\mathsf{Commit}, \mathsf{id}_{P_j}, (E_{P_j}, \pi^1_{P_j}))$ and all other parties input $(\mathsf{Commit}, \mathsf{id}_{P_j}, \perp)$
4. For $j = 1, \ldots, t$
   - The parties execute $\mathcal{F}_{\mathsf{Commit}}^{P_j}$ with input $(\mathsf{Open}, \mathsf{id}_{P_j})$ and abort if $\mathcal{F}_{\mathsf{Commit}}^{P_j}$ returns abort.
   - For all $P_j \neq P_i$ party P_j executes $\mathsf{ZK}.V((E_0, E_{P_i}), \pi^1_{P_i})$ and aborts if the verification algorithm fails.
5. $E^0 \leftarrow E_0$.
6. For $j = 1, \ldots, t$ do
   - Party P_j computes $E^j \leftarrow [s_{P_j}]E^{j-1}$.
   - $\pi^2_P \leftarrow \mathsf{ZK}.P((E_0, E_{P_j}, E^{j-1}, E^j), s_{P_j})$.
   - Broadcast $(E^j, \pi^2_{P_j})$ to all players.
   - All players execute $\mathsf{ZK}.V((E_0, E_{P_j}, E^{j-1}, E^j), \pi^2_{P_j})$ and abort if the verification algorithm fails.
7. Return E^t.

Figure 8. Group Action Computation for a Full Threshold Secret Sharing

Note that $\mathsf{GrpAction}(E_0, Q, [s])$ requires two zero-knowledge proofs of two isogenies to be computed per player. And each player needs to verify $2 \cdot (|Q| - 1)$ zero-knowledge proofs. However, the latency is $O(|Q|)$ due to the second loop.

If s is shared by our replicated scheme $\langle s \rangle$ we can use $\mathsf{GrpAction}(E, Q, [s])$, for a qualified set Q, to compute $\langle s \rangle E_0$ as well. The resulting operation we will denote by $\mathsf{GrpAction}(E_0, Q, \langle s \rangle)$. The modifications needed are as follows: Recall we have $s = \sum_{B \in \mathcal{B}} s_B$, and for a qualified set Q we can assign each s_B to a given player P via the function $\Psi_Q(B)$. Thus we can represent $\langle s \rangle$ as a full threshold scheme over the players in Q, where potentially each player plays the part of a set of players. Then we can execute the protocol as above, except that in line 4 we perform an additional check in that if $P' \in B$ then player P' checks whether $E_P = [s_P]E_0$. This check is performed by all players in $\mathcal{P}$, including those not in Q. This copes with the situations where $\mathcal{B}_M \neq \emptyset$, and we need to check consistency of the sharing.

Note, there is a trival optimization of the protocol for $\mathsf{GrpAction}(E_0, Q, \langle s \rangle)$ which does not expand the number of players artificially to $|\mathcal{B}|$ but keeps it at size $|Q|$. However, the above (less efficient) variant is what we will require for our protocol.

4.1 The Distributed Key Generation and Signing Protocols

We can now define our distributed key generation and signing protocols. The key generation protocol and the protocol to execute the signing operation in a distributed manner are given in Figure 9. The protocols are defined in the $(\mathcal{F}_{\mathsf{Rand}}, \mathcal{F}_{\mathsf{Commit}})$-hybrid models.

The Distributed Key Generation and Signing Protocols Π_{KeyGen}, Π_{Sign}

KeyGen: To generate a distributed key we execute:

1. Call $\mathcal{F}_{\mathsf{Rand}}.\mathsf{Init}()$.
2. For $i \in [1, \ldots, S-1]$ do
   (a) $\langle a_i \rangle \leftarrow \mathcal{F}_{\mathsf{Rand}}.\mathsf{PRSS}()$.
   (b) $E_i \leftarrow \mathsf{GrpAction}(E_0, Q, \langle a_i \rangle)$ for some qualified set Q. If this protocol aborts, then abort.
3. Output $\langle a_1 \rangle, \ldots, \langle a_{S-1} \rangle$ and $E_1, \ldots, E_{S-1}$.

Sign$(m, \langle s \rangle)$**:** For a set of qualified parties Q to sign a message m they execute:

1. Write $Q = \{P_1, \ldots, P_t\} \subset \mathcal{P}$.
2. For $i = 1, \ldots, t_{\mathsf{S}}$
   (a) Party P_j generates $b_{i,j} \leftarrow \mathbb{Z}/N\mathbb{Z}$, so as to form a full threshold sharing $[b_i]$ over the t parties.
   (b) The parties execute $E'_i \leftarrow \mathsf{GrpAction}(E_0, Q, [b_i])$.
3. The parties locally compute $(c_1, \ldots, c_{t_{\mathsf{S}}}) \leftarrow H(E'_1 \| \ldots \| E'_{t_{\mathsf{S}}} \| m)$.
4. For $i = 1, \ldots, t_{\mathsf{S}}$ party P_j computes $r_{i,j} \leftarrow b_{i,j} - \mathsf{sign}(c_i) \cdot \sum_{\Psi_Q(B)=P_j} a_{|c_i|,B}$.
5. The parties broadcast their values $r_{i,j}$ and locally compute $r_i \leftarrow \sum_{j=1}^{t} r_{i,j}$.
6. Output $\{(r_i, c_i)\}_{i=1}^{t_{\mathsf{S}}}$.

Figure 9. The Distributed Key Generation and Signing Protocols Π_{KeyGen}, Π_{Sign}

To estimate the cost of signing we use the estimate of 40ms from [BKV19] to compute a single isogency calculation $[b]E$ for a random $b \in \mathbb{Z}/N\mathbb{Z}$. By counting the number of such operations we can determine an approximate value for the execution time of our distributed signing protocol. The main cost is in computing $E' \leftarrow \mathsf{GrpAction}(E_0, Q, [b])$ a total of t_{S} times. We estimate the cost in terms of the number of parties $t = |Q|$ in the qualified set Q. Note, that one of the zero-knowledge proofs executed in Step Step 6 is $\mathsf{Special}$, whereas all others in this step are $\mathsf{General}$. Due to the sequential nature of the calculation this will have a latency of approximately $|Q| \cdot (1 + t_{\mathsf{ZK}}^{\mathsf{Special}})$ isogeny calculations for Step 3, $|Q| \cdot t_{\mathsf{ZK}}^{\mathsf{Special}}$ isogeny calculations for Step 4, and

$\left(1 + 4 \cdot \left((|Q| - 1) \cdot t_{\mathsf{ZK}}^{\mathsf{General}} + t_{\mathsf{ZK}}^{\mathsf{Special}}\right)\right)$ isogeny calculations for Step 6, of Figure 8. Thus the *rough* total execution time is about

$$t_{\mathsf{S}} \cdot \left(|Q| \cdot (1 + 2 \cdot \mathsf{ZK}^{\mathsf{Special}} + 4 \cdot \mathsf{ZK}^{\mathsf{General}}) + 4 \cdot \mathsf{ZK}^{\mathsf{Special}} - 4 \cdot \mathsf{ZK}^{\mathsf{General}} + 1\right)$$

isogeny calculations.

Taking the specimen parameters of $t_{\mathsf{ZK}}^{\mathsf{General}} = 112$ and $t_{\mathsf{ZK}}^{\mathsf{Special}} = 70$ and $t_{\mathsf{S}} = 7$, and considering the case of a set Q with two members, this gives a latency of about $7 \cdot (2 \cdot (1 + 2 \cdot 70 + 4 \cdot 112) + 4 \cdot 70 - 4 \cdot 112 + 1) \cdot 0.040 = 283$ seconds per party. Which is just under five minutes per party.

4.2 Proofs of Security

To prove the distributed key generation and signing protocols secure we present a simulation of the environment to the adversary. The simulator has access to a signing functionality for some unknown secret key, via the functionality in Figure 5. For securtiy to hold the adversary must not be able to distinguish between executing in the real environment and executing with the simulation. Our simulation requires rewinding of the adversary in order to extract the witnesses for the associated zero-knowledge proofs. Thus our security proof does not provide a UC-proof of the security of the protocol. Thus our protocol should only be considered 'stand-alone' secure.

KeyGen Simulation: The simulator first calls the functionality $\mathcal{F}_{\mathsf{DSign}}$, which outputs a replicated secret sharing of the associated secret keys $\langle a_i \rangle$ for the adversary, i.e. the simulator learns the values $a'_{i,B}$ for all $B \in \mathcal{B}_A \cup \mathcal{B}_M$, but not for those values in $\mathcal{B}_H$. The simulator now passes the values $a'_{i,B}$ for $B \in \mathcal{B}_A \cup \mathcal{B}_M$ to the adversary by simulating the $\mathcal{F}_{\mathsf{Rand}}.\mathsf{PRSS}()$ protocol.

For each $i \in [1, \ldots, S-1]$ the adversary now engages in an execution of $\mathsf{GrpAction}(E_0, Q, \langle a_i \rangle)$; note E_0 is fixed across all public keys and hence known to the simulator ahead of time. From the committed zero-knowledge proofs π_P^1 the simulator is able to extract the value $a_{i,B}$ enterred by the adversary in the first round of proofs. Note, this value may be different from the values returned by the PRSS, but that is allowed in our security model, as long as it does not contradict a value corresponding to an element in $\mathcal{B}_M$ (if there is a contradiction we will be able to abort later when the real system will abort during the check for this fact). The extracted values $a_{i,B}$ are now passed to the functionality, which completes them to a valid set of shares of the secrets and returns the corresponding public key $E_0, \ldots, E_{S-1}$.

The simulator picks a single honest sharing $B^* \in \mathcal{B}_H$ and generates $a_{i,B}$ for $B \in \mathcal{B}_H \setminus \{B^*\}$ at random. Thus a_{i,B^*} will be the secret key values which are unknown to the simulator. We let j denote the player index corresponding to the element B^*. We let the curve E_{P_j} in Step 3 of Figure 8 denote a random element of the isogeny graph. We can now fake the associated zero-knowledge proof $\pi_{P_j}^1$ using the simulator for the zero-knowledge proof The commitments can now be opened.

Now look at Step 6 of Figure 8. All steps for honest players can be simulated exactly by following the real protocol, bar that for the party P_j which holds the unknown share $a_{i,B*}$. The input to this party in execution i will be

$$E_i^{j-1} = \left[\sum_{k=1}^{j-1} s_{P_k}\right] E_0,$$

whilst the output needs to be

$$E_i^j = \left[- \sum_{k=j+1}^{t} s_{P_k} \right] E_i,$$

so as to create the correct output public key E_i. The value E_i^j can thus be computed by the simulator in Step 6 of Figure 8, and the associated zero-knowledge proof can hence be simulated as well.

If the adversary deviates from the protocol in any way, bar changing the values of $a_{i,B}$ for $B \in \mathcal{B}_A$ in the first phase, this is caught be the zero-knowledge proofs and the simulator will be able to abort. Thus the protocol, assuming no abort occurs, will output the same public key as provided by the ideal functionality.

Sign Simulation: The signing simulation is roughly the same as the key generation simulation. For the qualified set Q, the adversarial inputs can be derived from the initial commitments in $\mathsf{GrpAction}(E_0, Q, [b])$. We let j now be the player for which $\Psi_Q(B*) = P_j$. In our simulation of $\mathsf{GrpAction}(E_0, Q, [b])$ we can defined b_i for $P_i \in \mathcal{B}_H \setminus \{P_j\}$ at random, leaving the value b_j unknown and 'fixed' by the implicit equation given by the signature (r, c) returned by the functionality which gives us $E' = [b]E_0 = [r]E_c$.

The final part of the signature which needs simulating is the output of the r_i for the honest players in Q. For $i \neq j$ this is done exactly as one would in the real protocol. For party j, we know what the adversary *should* output and hence can define $r_j = r - \sum_{i \neq j} r_i$.

If the adversary deviates from the protocol in the final step, and uses an invalid value of r_i. Then the adversary will learn the signature, but the honest players will abort; which is exactly the behaviour required by the ideal functionality.

Acknowledgments

We would like to thank Frederik Vercauteren for the numerous and useful discussions on the arithmetic of isogenies. This work has been supported in part by ERC Advanced Grant ERC-2015-AdG-IMPaCT, by the FWO under an Odysseus project GOH9718N and by CyberSecurity Research Flanders with reference number VR20192203. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the ERC or FWO.

References

BDV19. Luis T. A. N. Brandao, Michael Davidson, and Apostol Vassilev. NIST 8214A (Draft): Towards NIST standards for threshold schemes for cryptographic primitives: A preliminary roadmap, 2019.

BKV19. Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. CSI-FiSh: Efficient isogeny based signatures through class group computations. *IACR Cryptology ePrint Archive*, 2019:498, 2019.

CDI05. Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 342–362. Springer, Heidelberg, February 2005.

CDK+18. Benoît Cogliati, Yevgeniy Dodis, Jonathan Katz, Jooyoung Lee, John P. Steinberger, Aishwarya Thiruvengadam, and Zhe Zhang. Provable security of (tweakable) block ciphers based on substitution-permutation networks. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 722–753. Springer, Heidelberg, August 2018.

CLM+18. Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: An efficient post-quantum commutative group action. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 395–427. Springer, Heidelberg, December 2018.

Cou06. Jean-Marc Couveignes. Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291, 2006. http://eprint.iacr.org/2006/291.

CS19. Daniele Cozzo and Nigel P. Smart. Sharing the LUOV: Threshold post-quantum signatures. *IACR Cryptology ePrint Archive*, 2019:1060, 2019.

DG19. Luca De Feo and Steven D. Galbraith. SeaSign: Compact isogeny signatures from class group actions. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 759–789. Springer, Heidelberg, May 2019.

DK01. Ivan Damgård and Maciej Koprowski. Practical threshold RSA signatures without a trusted dealer. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 152–165. Springer, Heidelberg, May 2001.

DKLs18. Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Secure two-party threshold ECDSA from ECDSA assumptions. In *2018 IEEE Symposium on Security and Privacy*, pages 980–997. IEEE Computer Society Press, May 2018.

DPV19. Thomas Decru, Lorenz Panny, and Frederik Vercauteren. Faster SeaSign signatures through improved rejection sampling. In Jintai Ding and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019*, pages 271–285. Springer, Heidelberg, 2019.

FM19. Luca De Feo and Michael Meyer. Threshold schemes from isogeny assumptions. *IACR Cryptology ePrint Archive*, 2019:1288, 2019.

GG18. Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1179–1194. ACM Press, October 2018.

GGN16. Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *ACNS 16*, volume 9696 of *LNCS*, pages 156–174. Springer, Heidelberg, June 2016.

GJKR96. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold DSS signatures. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 354–371. Springer, Heidelberg, May 1996.

Lin17. Yehuda Lindell. Fast secure two-party ECDSA signing. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 613–644. Springer, Heidelberg, August 2017.

LN18. Yehuda Lindell and Ariel Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1837–1854. ACM Press, October 2018.

LNR18. Yehuda Lindell, Ariel Nof, and Samuel Ranellucci. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. *IACR Cryptology ePrint Archive*, 2018:987, 2018.

MR01. Philip D. MacKenzie and Michael K. Reiter. Two-party generation of DSA signatures. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 137–154. Springer, Heidelberg, August 2001.

RS06. Alexander Rostovtsev and Anton Stolbunov. Public-Key Cryptosystem Based On Isogenies. Cryptology ePrint Archive, Report 2006/145, 2006. http://eprint.iacr.org/2006/145.

Sho00. Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 207–220. Springer, Heidelberg, May 2000.

Sto12. Anton Stolbunov. *Cryptographic schemes based on isogenies*. PhD thesis, NTNU, 2012.

Vé71. Jacques Vélu. Isogènies entre courbes elliptiques. *C.R. Acad. Sc. Paris, Série A.*, 273:238–241, 1971.