CSI-SharK: CSI-FiSh with Sharing-friendly Keys

Shahla Atapoor [1], Karim Baghery [1], Daniele Cozzo [2,1], and Robi Pedersen [1]

[1] imec-COSIC, KU Leuven, Leuven, Belgium
[2] IMDEA Software Institute, Madrid, Spain
firstname.lastname@kuleuven.be,
daniele.cozzo@imdea.org

Abstract. CSI-FiSh is one of the most efficient isogeny-based signature schemes, which is proven to be secure in the Quantum Random Oracle Model (QROM). However, there is a bottleneck in CSI-FiSh in the threshold setting, which is that its public key needs to be generated by using $k-1$ secret keys. This leads to very inefficient threshold key generation protocols and also forces the parties to store $k-1$ secret shares. We present CSI-SharK, a new variant of **CSI**-FiSh that has more **Shar**ing-friendly **K**eys and is as efficient as the original scheme. This is accomplished by modifying the public key of the ID protocol, used in the original CSI-FiSh, to the equal length Structured Public Key (SPK), generated by a *single* secret key, and then proving that the modified ID protocol and the resulting signature scheme remain secure in the QROM. We translate existing CSI-FiSh-based threshold signatures and Distributed Key Generation (DKG) protocols to the CSI-SharK setting. We find that DKG schemes based on CSI-SharK outperform the state-of-the-art actively secure DKG protocols from the literature by a factor of about 3, while also strongly reducing the communication cost between the parties. We also uncover and discuss a flaw in the key generation of the actively secure CSI-FiSh based threshold signature *Sashimi*, that can prevent parties from signing. Finally, we discuss how (distributed) key generation and signature schemes in the isogeny setting are strongly parallelizable and we show that by using C independent CPU threads, the total runtime of such schemes can basically be reduced by a factor C. As multiple threads are standard in modern CPU architecture, this parallelizability is a strong incentive towards using isogeny-based (distributed) key generation and signature schemes in practical scenarios.

Keywords: Isogeny-based cryptography · DKG · Threshold Schemes · CSIDH

1 Introduction

Following Shor's attack [Sho94] on the Factoring and Discrete Logarithm (DL) problems, researchers started exploring post-quantum cryptographic techniques

to construct primitives and protocols that can be secure in the presence of quantum adversaries. One of these areas is isogeny-based cryptography, which was initially proposed by Couveignes [Cou06] and by Rostovtsev and Stolbunov [RS06, Sto10]. These proposals use hardness assumptions based on isogenies between ordinary elliptic curves, but were strongly weakened by the quantum attack of Childs, Jao and Soukarev [CJS14] a few years later. Later proposals by Jao and De Feo [JD11] (SIDH) and Castryck et al. [CLM+18] (CSIDH) use supersingular elliptic curves instead of ordinary ones. While the attack by Childs et al. is still applicable to the latter, the authors of [CLM+18] managed to push the secure parameter set back into practical sizes. SIDH on the other hand was recently broken by a different attack, designed by Castryck and Decru [CD22] and further improved in subsequent works [MM22, Rob22]. These attacks do not apply to CSIDH-based schemes.

Based on CSIDH, two signature schemes called SeaSign [DG19] and CSI-FiSh [BKV19] have been proposed. The basic version of both these signature schemes are based on the same ID protocol that was introduced in [RS06] and both papers discuss extending the public keys in order to reduce the soundness error rate by using multiple secret keys. To allow this extension, the underlying language in the ID protocol is changed, and a witness for the Multi-Target Group Action Inverse Problem (MT-GAIP) [DG19,BKV19] is proven. The fundamental difference between SeaSign and CSI-FiSh is that in the latter, the authors have performed a record computation of the underlying ideal-class group, which allows uniform sampling and canonical representations of ideals. As a result, unlike SeaSign, CSI-FiSh does not need to rely on rejection sampling and was therefore one of the first practical signature scheme based on isogenies, with signing times just below 3 seconds in the basic version of the scheme and a signature size of 1.8 KB. With an extended public key, signing can even be done in a few hundred milliseconds (Table 3 in [BKV19]). Other practical isogeny-based signature schemes include [GPS17, YAJ+17] and later [DKL+20]. Recently, Baghery, Cozzo and Pedersen [BCP21] presented a new extension of the basic ID protocol used in CSI-FiSh. They extend the public key to a *Structured Public Key* (SPK), which is generated using a single secret key and a set of public integers. This allows working with a larger challenge space and efficiently proving the actual witness, rather than the differences of two curves, as in MT-GAIP. However, to extract the single secret key, they require a Trusted Third Party (TTP) to generate the keys. Furthermore, to prove the security of their ID protocol, they rely on a new computational assumption, so-called $\mathbb{C}_k$-Vectorization Problem with Auxiliary Inputs ($\mathbb{C}_k$-VPwAI, see Definition 3.2), which its hardness recently was analysed in [DHK+23].

By virtue of its flexibility, CSI-FiSh is used as the base scheme to build various isogeny-based protocols, such as threshold signatures [DM20, CS20], lossy signatures [EKP20], and forward secure ones [SD21]. Due to a wide range of applications (e.g. in blockchain), Threshold Signature Schemes (TSSs) have received more attention in recent years. Such schemes allow distributing the secret key into shares among multiple parties or devices, such that only a set of au-

thorized parties can jointly sign a message to produce a single signature. Key recovery attacks on TSSs require more effort than the non-threshold ones, as the adversary has to attack more than one device or party simultaneously. A large amount of effort has been put on constructing threshold versions of classic schemes based on discrete logarithms and RSA. Most of these schemes could be integrated with a Distributed Key Generation (DKG) protocol, which was initially proposed by Pedersen in [Ped92]. A secure version of it along with robustness guarantees was proposed by Gennaro et al. [GJKR07]. On the other hand, research on threshold isogeny-based signatures is very young. In 2020, De Feo and Meyer [DM20] proposed a threshold version of CSI-FiSh using Shamir secret sharing. The protocol is almost as efficient as a CSI-FiSh signature but only passively secure. Cozzo and Smart [CS20] then proposed Sashimi, the first actively secure threshold signature scheme with abort based on CSI-FiSh, that uses replicated secret sharing. Sashimi uses Zero-Knowledge (ZK) proofs to achieve active security, resulting in the rather elevated signing times of around 5 minutes for two parties. The cost of their DKG step is even higher, but was not estimated by the authors. Later, Beullens et al. proposed CSI-RAShi [BDPV21], an actively secure DKG protocol for CSI-FiSh using Shamir secret sharing. By introducing a new primitive called piecewise verifiable proofs, the authors manage to reduce the cost of generating public keys to about 18 seconds per party. Still, Sashimi [CS20] as well as CSI-RAShi [BDPV21], when used for extended public keys, remain quite inefficient to be used in practice.

1.1 Our Contributions and Techniques

CSI-SharK Signature Scheme. As our initial contribution, we present *CSI-SharK* in Section 3, a new variant of **CSI**-FiSh [BKV19] that has more **Shar**ing-friendly **K**eys. To this end, we slightly modify the ID protocol used in CSI-FiSh [BKV19] and then prove that the modified ID protocol and the resulting signature scheme remain secure in the QROM. Our main modification is to change the public key of CSI-FiSh to a SPK, $(E_0, E_1 = [c_1 \cdot \mathsf{sk}]E_0, \cdots, E_{k-1} = [c_{k-1} \cdot \mathsf{sk}]E_0)$, which is generated using a *single* secret key sk and $k-1$ public integers $c_1, \cdots, c_{k-1}$, instead of using $k-1$ distinct secret keys. Then we prove that the modified ID protocol is special sound for the language of MT-GAIP, and Honest-Verifier Zero-Knowledge (HVZK) for the language of $\mathbb{C}_k$-VPwAI. Roughly speaking, the modified ID protocol achieves to the best of the two ID protocols presented in [BKV19] and [BCP21]. In comparison with [BKV19], the modified ID protocol has only a single secret key but needs to rely on an additional hardness assumption. In comparison with the one proposed in [BCP21], it does not require a TTP, but only proves knowledge of a witness for the MT-GAIP, which is sufficient for building a signature scheme. We turn the modified ID protocol into a signature scheme in the standard manner using the Fiat–Shamir transform [FS87]. Our results and analysis show that, due to having a SPK and a single secret key, CSI-SharK is a suitable replacement for CSI-FiSh and achieves the same efficiency as the original scheme, without the need of storing multiple secret keys. The real advantage of CSI-SharK however manifests itself when it is

used to build threshold schemes, which can be much more efficient than in the CSI-FiSh setting.

Threshold Schemes. As a next contribution, we show how to distribute key generation schemes and signatures using structured public keys. To this end, we first review the state-of-the-art DKG schemes, i.e. the passively secure scheme by De Feo and Meyer [DM20], as well as the actively secure Sashimi [CS20] and CSI-RAShi [BDPV21]. We notice that there is a flaw in the current description of the DKG protocol in Sashimi that can lead to an attack, which could allow an adversary to prevent an honest party from using its share to sign. We then describe a version of Sashimi that we use as an actively secure full-threshold signature scheme with abort, preventing the attack. Finally, we discuss optimizations of these state-of-the-art schemes to act as a baseline for comparison with our proposed schemes. Then, in Section 5, we show how using structured public keys decreases the overall computational and communication cost of DKG protocols. We then describe ways to further reduce the computational cost of these protocols. The final protocols have a computational cost decreased by about a factor 3, when compared to the most optimal implementations without a SPK, while furthermore reducing the communication cost between the different parties by a significant factor, which depends on the public key size. In Appendix A, we also discuss how to translate the signatures from [DM20, CS20, BDPV21] to our setting. We note that the computational cost of the signing algorithm is not decreased by using SPKs, but generally threshold protocols based on CSI-

Table 1. Comparison of our proposed CSI-SharK-based key generation schemes and the existing ones based on CSI-FiSh, for n parties, a public key with $k-1$ elliptic curves and a security parameter λ. Next to displaying the number of necessary secret sharings (Sec. Shar.), we present the leading term in the computational cost of the public key generation (PKGen) in terms of group actions as well as the leading term in communication cost per party (Comm.) in bits (we assume $k \gg n$). The latter two mainly depend on the parameters k, n and λ, as well as $\log p$, the size of the underlying prime field $\mathbb{F}_p$. We further display the hardness assumptions underlying the security of the different schemes. These assumptions are formally introduced in Sec. 2 and Sec. 3. TTP: Trusted Third Party, DKG: Distributed Key Generation, SSS: Shamir Secret Sharing, FT: Full-Threshold secret sharing.

	(Distributed) Key Generation Schemes from Isogenies					
	[DM20]	Section 5.1	Sashimi	Section 5.2	CSI-RAShi	Section 5.3
Adversary:	Passive	Passive	Active	Active	Active	Active
Setup by:	TTP	TTP	DKG	DKG	DKG	DKG
Sec. Shar.:	$k-1$ SSS	1 SSS	$k-1$ FT	1 FT	$k-1$ SSS	1 SSS
PKGen:	$k-1$	$k-1$	$5kn\lambda$	$n(k+2\sqrt{k})\lambda$	$4kn\lambda$	$n(k+2\sqrt{k})\lambda$
Comm.:	$\frac{1}{2}k\log p$	$\frac{1}{2}\log p$	$k\lambda\log p$	$(k+\frac{1}{2}\sqrt{k}\lambda)\log p$	$k\lambda(8n+\log p)$	$(k+\frac{1}{2}\sqrt{k}\lambda)\log p$
Hardness Assumpt.:	MT-GAIP, P-DDHA	MT-GAIP, P-DDHA, $\mathbb{C}_k$-VPwAI	MT-GAIP, dGAIP	MT-GAIP, dGAIP, $\mathbb{C}_k$-VPwAI	MT-GAIP, dGAIP	MT-GAIP, $\mathbb{C}_k$-dGAIP, $\mathbb{C}_k$-VPwAI

SharK (e.g. threshold signatures) are more efficient and we also discuss some optimizations that are applicable to both settings.

Parallel Computations in Isogeny-based Schemes. Due to the round-robin computational structure in isogeny-based threshold schemes, the total latency of distributed protocols can be huge. Because of this issue, based on the estimations done in Sashimi [CS20], signing could take about 5 minutes with two parties. As the final contribution, presented in Section 6, we observe that many of the computations in both non-threshold and threshold signatures built from isogenies are parallelizable. In light of this observation, we propose a general parallel execution strategy to be used in the cases that *one* (as in CSI-FiSh or CSI-SharK) or *multiple* parties (as in TSSs or DKG protocols) need to compute multiple independent elliptic curves, e.g. $[x_1]E_0, \ldots, [x_r]E_0$. For the single party case, the computation simply uses the different cores (or threads) of a multi-core CPU to compute a subset of curves, while in the multi-party case, different parties further can run their computations in parallel. Using the proposed strategy and several optimizations, we could significantly reduce the runtimes of CSI-SharK, CSI-FiSh, and their threshold variants, making them quite practical. We give an overview of the overall complexities in Table 1.

2 Preliminaries

In this section, we briefly review secret sharing schemes, commitment schemes and isogeny-based cryptography. For an introduction to sigma protocol and (threshold) signatures, we refer the reader to Appendices A.1 and A.2.

Notation. We let $x \leftarrow X$ denote the uniformly random assignment to the variable x from the set X, assuming a uniform distribution over X. 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 $\mathcal{D}$. We further write $\mathbb{Z}_N = \mathbb{Z}/N\mathbb{Z}$ and define $\log(x) := \log_2(x)$.

Secret sharing schemes. We define an (n, t)-threshold access structure as a set consisting of n players, where each player holds shares of a secret, so that any subset of t or more players can reconstruct this secret. We assume that all players are Probabilistic Polynomial Time (PPT) Turing machines. We revisit two well known ways for realizing a (n, t)-threshold access structure that we will use in our distributed signature schemes, namely the Shamir Secret Sharing (SSS) scheme [Sha79] and the Full-Threshold Secret Sharing (Full-TSS) scheme. Unlike the traditional versions that are defined over a prime field, we will use them over an integer ring $\mathbb{Z}_N$.

Shamir Secret Sharing. In (n, t)-Shamir secret sharing, a common polynomial $f(x) \in \mathbb{Z}_N[x]$ of degree $t - 1$ is chosen, such that the secret s is set to be its constant term, i.e. $s = f(0)$. Each party P_i for $i \in \{1, \cdots, n\}$ is assigned the

Init: Given (Init, P_i, B) from all parties, this initializes a commitment functionality from party P_i to the parties in B. This is shown with $\mathcal{F}_{\mathsf{Commit}}^{i,B}$, if B is a singleton set $B = \{j\}$ then we write $\mathcal{F}_{\mathsf{Commit}}^{i,j}$, and if $B = \mathcal{P} \setminus \{i\}$ then we write $\mathcal{F}_{\mathsf{Commit}}^{P_i}$.
Commit: On input of $(\mathsf{Commit}, \mathsf{id}, \mathsf{data})$ from parties P_i and $(\mathsf{Commit}, \mathsf{id}, \bot)$ from all parties in B the functionality stores $(\mathsf{id}, \bot)$.
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.

Fig. 1. The Functionality $\mathcal{F}_{\mathsf{Commit}}$

secret share $s_i = f(i)$ Then any subset $S \subset \{1, \ldots, n\}$ of at least t parties can reconstruct the secret s via Lagrange interpolation by computing $s = f(0) = \sum_{i \in S} s_i \cdot L_{0,i}^S$, where

$$L_{0,i}^S := \prod_{j \in S \setminus \{i\}} \frac{j}{j-i} \pmod N,$$

are Lagrange basis polynomials evaluated at 0. Any subset of less than t parties are not able to find $s = f(0)$, as this is information-theoretically hidden, even given $t-1$ shares. Since we will be working over the ring $\mathbb{Z}_N$ with N composite, the difference $j-i$ of any two elements in $i, j \in S$ must be invertible modulo N. If q is the smallest prime factor of N, it is enough to require that $n < q$.

Full-threshold secret sharing. In a Full-TSS, a secret $s \in \mathbb{Z}_N$ is additively shared among $P_1, \ldots, P_n$. Specifically, each party P_i holds a random share $s_i \in \mathbb{Z}_N$ such that $s = s_1 + \cdots + s_n \bmod N$. Clearly, all parties are required to recover the secret s and thus a Full-TSS realizes a (n, n)-threshold access structure. Since the shares are random, up to $n-1$ parties cannot get any information about the secret s, as the remaining share information-theoretically hides it.

Commitment Schemes. In our protocols, we assume parties have access to a commitment functionality $\mathcal{F}_{\mathsf{Commit}}$, which allows one party to commit, and later open the value to a set of parties. We assume the opened value is only available to the targeted receivers and is sent over a secure communication channel. The description of $\mathcal{F}_{\mathsf{Commit}}$ is provided in Figure 1, which can be easily implemented in the random oracle model. This gives the commitment properties such as extractability and equivocability that are crucial for our security proofs.

Isogeny-based cryptography. Isogenies are rational maps between two elliptic curves $\varphi : E \to E'$, that are also homomorphisms with respect to the natural group structure of E and E' [DF17, Sil09]. For simplicity, we introduce isogenies as an abstract cryptographic concept in this section and focus only on the high-level ideas that allow us to build cryptographic protocols on top of them. For a more thorough introduction, we refer the reader to [CLM+18] and [DF17].

Isogeny computations can be abstracted as a group $\mathcal{G}$ acting freely and transitively on a set $\mathcal{E}$ [Cou06]. In the case relevant to this work, $\mathcal{E}$ denotes the set

of supersingular elliptic curves over a finite field $\mathbb{F}_p$ where p is a large prime. $\mathcal{G}$ denotes the ideal-class group $\mathrm{Cl}(\mathcal{O})$, where $\mathcal{O}$ is an order of the quadratic imaginary field $\mathbb{Q}(\sqrt{-p})$, isomorphic to the $\mathbb{F}_p$-rational endomorphism ring $\mathsf{End}_{\mathbb{F}_p}(E)$ elliptic curves $E \in \mathcal{E}$. Isogenies are uniquely defined through the kernels of the ideals in $\mathcal{O}$. Throughout this work, we assume that the class group, as well as a generating ideal $\mathfrak{g}$ of it are known. In that case, we can define this group action as $[\cdot] : \mathbb{Z}_N \times \mathcal{E} \to \mathcal{E}$, where N is the size of the class group; asymptotically $N \approx \sqrt{p}$ [CLM+18,Sie35]. As an example for the group action, we could identify the isogeny $\varphi : E \to E'$ with some element $a \in \mathbb{Z}_N$, such that $E' = [a]E$ holds. This means that the separable part of the isogeny φ has the kernel $\bigcap_{\alpha \in \mathfrak{g}^a} \ker \alpha$. Note that $[a]([b]E) = [a+b]E$. In cryptographic settings, this group action comes equipped with some hardness assumptions, that are also difficult for quantum computers.

Definition 2.1 (Group action inverse problem (GAIP) [CLM+18,DG19]). *Given two supersingular elliptic curves $E, E' \in \mathcal{E}$ over the same finite field $\mathbb{F}_p$ and with $\mathsf{End}_{\mathbb{F}_p}(E) \simeq \mathsf{End}_{\mathbb{F}_p}(E') \simeq \mathcal{O}$, find $a \in \mathbb{Z}_N$, such that $E' = [a]E$.*

Definition 2.2 (Decisional GAIP [Sto12,BDPV21]). *Given a triple $([a]E, [b]E, [c]E)$, where $E \in \mathcal{E}$, with the same $\mathbb{F}_p$-rational endomorphism ring, decide whether $[c]E = [a+b]E$ or not.*

Isogenies are efficiently computable, but hard to invert. As such, they are a powerful tool to build post-quantum cryptographic protocols. One of these schemes is the isogeny-based signature scheme CSI-FiSh proposed by Beullens, Kleinjung and Vercauteren [BKV19]. The basic version of CSI-FiSh is based on an ID protocol with binary challenges initially proposed in [Sto12]. CSI-FiSh starts with the special elliptic curve $E_0 : y^2 = x^3 + x$. Public keys are created with the action of an element $a \in \mathbb{Z}_N$ on E_0. The owner of the public key $E = [a]E_0$ proves knowledge of the secret key a by a standard binary sigma protocol. This is then turned into a signature scheme using the Fiat–Shamir heuristic [FS87]. Note that the class group enjoys a symmetry around the elliptic curve E_0, as the elliptic curve $[-a]E_0$ is $\mathbb{F}_p$-isomorphic to the quadratic twist of $[a]E_0$, a map that is easily computable without any extra information. As a result, we can implicitly include the twist in the public key and extend the challenge space to $\{-1, 0, 1\}$, so that the soundness error rate is reduced to $\frac{1}{3}$.

To further reduce the number of repetitions needed to achieve a security level of 2^{sec}, the authors of [BKV19] used a technique proposed in [DG19] and enlarged the public key by using multiple secret keys $(a_1, \ldots, a_{k-1})$ and an extended public-key of the $k-1$ associated elements. The resulting number of repetitions is $t_S = \lceil \mathsf{sec} \log_{2k-1} 2 \rceil$, although one can reduce t_S a little bit by choosing a 'slow' hash function. We present the CSI-FiSh in Figure 2. Note that $\mathsf{H} : \{0,1\}^\star \to \{-(k-1), \ldots, k-1\}^{t_S \lceil \log(2k-1) \rceil}$ represents a hash function modeled as a random oracle. We simply denote the twist of a curve E_a as E_{-a}. The extension of the public key in this way leads to a change in the underlying language, and the prover now must prove that it knows a secret $s \in \mathbb{Z}_N$ such that $E_j = [s]E_i$ for

$\mathsf{KeyGen}(1^{\mathsf{sec}})$**:** Given E_0, the secret and public key are generated as follows:

1. For $i = 1, \ldots, k-1$, sample $a_i \leftarrow \mathbb{Z}_N$ and compute $E_i = [a_i]E_0$.
2. Return $\mathsf{sk} = (a_1 \ldots, a_{k-1})$, $\mathsf{pk} = (E_0, \ldots, E_{k-1})$.

$\mathsf{Sign}(\mathsf{sk}, m)$**:** To sign a message m, the signer performs

1. For $i = 1, \ldots, t_S$: sample $b_i \leftarrow \mathbb{Z}_N$, and set $E_{b_i} = [b_i]E_0$.
2. Set $(d_1, \ldots, d_{t_S}) = \mathsf{H}(E_{b_1}, \ldots, E_{b_{t_S}} \parallel m)$.
3. For $i = 1, \ldots, t_S$: set $r_i = b_i - \mathrm{sign}(d_i)a_{|d_i|} \pmod N$.
4. Return $\{(r_i, d_i)\}_{i=1}^{t_S}$.

$\mathsf{Verify}(\{(r_i, d_i)\}_{i=1}^{t_S}, m, \mathsf{pk})$**:** To verify a signature $\{(r_i, d_i)\}_{i=1}^{t_S}$ on m, one performs:

1. For $i = 1, \ldots, t_S$: compute $E'_{b_i} = [r_i]E_{d_i}$.
2. $(d'_1, \ldots, d'_{t_S}) = \mathsf{H}(E'_{b_1}, \ldots, E'_{b_{t_S}} \| m)$.
3. If $(d_1, \ldots, d_{t_S}) = (d'_1, \ldots, d'_{t_S})$ then return valid, else output $\mathsf{invalid}$.

Fig. 2. CSI-FiSh Signature Scheme [BKV19].

some pair of elliptic curves appearing in the public key list. As a result, CSI-FiSh relies on the hardness of a multi-target version of GAIP, called MT-GAIP.

Definition 2.3 (MT-GAIP [DG19,BKV19]). *Given k supersingular elliptic curves $E_0, E_1, \ldots, E_k \in \mathcal{E}$ over $\mathbb{F}_p$ with the same $\mathbb{F}_p$-rational endomorphism ring, find $a \in \mathbb{Z}_N$, such that $E_i = [a]E_j$ for some $i, j \in \{0, \ldots, k\}$ with $i \neq j$.*

CSI-FiSh is sEUF-CMA secure in the QROM under the MT-GAIP assumption [BKV19]. For later reference, we also introduce the Power-Decision Diffie-Hellman (Power-DDHA) assumption defined in [DM20], which is used in some of our schemes.

Definition 2.4 (Power-DDHA). *Given an element $E \in \mathcal{E}$ and $a \in \mathbb{Z}_N$. The a-Power-DDHA problem is: given $(a, E, [s]E, F)$, where s is uniformly sampled from $\mathbb{Z}_N$ and where $F \in \mathcal{E}$, either sampled from the uniform distribution on $\mathcal{E}$ or $F = [as]E$, decide which distribution F is drawn from.*

For details on the quantum security of CSIDH-based schemes, see [CLM+18, Pei20, BS20, CSCDJRH20].

3 CSI-FiSh with Structured Public Keys

In this section, we revisit a different way of extending the public key (of the ID protocol) used in CSI-FiSh, where instead of sampling a total of $k-1$ different secrets $a_1, \ldots, a_{k-1}$, we use $k-1$ different multiples of the same secret key a to generate the public key. The idea to build such an ID protocol for proving the knowledge of a was first proposed by Baghery et al. [BCP21], but needed a TTP to generate the public key in order to guarantee the correct structure. Such a *structured* public key (SPK) consists of an integer set $\mathbb{C}_{k-1} = \{c_0 = 0, c_1 = 1, c_2, \ldots, c_{k-1}\}$ and a set of elliptic curves[3] $\{E_i = [c_i a]E_0\}_{i=0,\ldots,k-1}$. In their ID

[3] For simplicity, we include E_0 in the public-key. Note that [0] simply denotes the neutral element of the group action.

protocol, to ensure extractability of the witness modulo a composite number N, the integer set $\mathbb{C}_{k-1}$ has to be an exceptional set [BCPS18, DLSV20], as defined below. When E_0 is the same special base curve as in CSI-FiSh, the authors further introduce the notation of superexceptional sets [BCP21]. The latter case is referred to as a *symmetric* hard homogeneous space.

Definition 3.1 ((Super-)Exceptional set). *An* exceptional set modulo N *is a set* $\mathbb{C}_{k-1} = \{c_0, \ldots, c_{k-1}\} \subseteq \mathbb{Z}_N$*, where the pairwise difference* $c_i - c_j$ *of all elements* $c_i \neq c_j$ *is invertible modulo* N*. A* superexceptional set modulo N *is an exceptional set* $\mathbb{C}_{k-1} = \{c_0, \ldots, c_{k-1}\}$*, where also the pairwise sum* $c_i + c_j$ *of all elements* c_i, c_j *(including* $c_i = c_j$*) is invertible modulo* N*.*

The hardness of obtaining the secret key from a structured public key relies on the following computational problem which is defined in [BCP21].

Definition 3.2 ($(c_0, c_1, \cdots, c_{k-1})$-Vectorization Problem with Auxiliary Inputs ($\mathbb{C}_{k-1}$-VPwAI)). *Given an element* $E \in \mathcal{E}$ *and the pairs* $(c_i, [c_i x]E)_{i=1}^{k-1}$*, where* $\mathbb{C}_{k-1} = \{c_0 = 0, c_1 = 1, c_2, \ldots, c_{k-1}\}$ *is an exceptional set, find* $x \in \mathbb{Z}_N$*.*

The ID protocol of Baghery et al. [BCP21] is designed to allow the prover to efficiently prove the knowledge of the secret key a, while relying on the fact that the public key indeed has the desired structure. The authors of [BCP21] solve this problem by letting either a TTP generate these keys, or alternatively by relying on computationally rather heavy *well-formedness proofs* for proving the correctness of the key generation. We refer to the original source for more details.

3.1 The Modified Identification Protocol

In this section, we construct an ID protocol that achieves the best of the two ID protocols constructed in [BCP21] and [BKV19]. The protocol of [BCP21] uses structured public keys and soundness relies on the $\mathbb{C}_{k-1}$-VPwAI (Definition 3.2), while the protocol from [BKV19] uses non-structured (extended) public keys and soundness relies on MT-GAIP. Note that in order to guarantee that the public keys in the former actually have the correct structure, the scheme either relies on trusted third parties generating the keys or on heavy proofs of *well-formedness*.

The idea behind our new ID protocol is to work with structured public keys, but nevertheless base the soundness of the protocol on MT-GAIP. The advantage of relying on MT-GAIP, rather than on $\mathbb{C}_{k-1}$-VPwAI for soundness, is that we do not need a TTP, or heavy proofs of well-formedness, to generate the public keys. The SPK thus becomes a perk for the prover rather than a requirement for the protocol. The idea is simple, yet powerful: the prover first samples a secret key $a \leftarrow \mathbb{Z}_N$ and a public *(super-)exceptional* set $\mathbb{C}_{k-1} = \{c_0 = 0, c_1 = 1, c_2, \cdots, c_{k-1}\}$, then generates and publishes the SPK $(E_0, E_1, \ldots, E_{k-1})$, where $E_i = [c_i a]E_0$ for $i = 1 \ldots, k-1$. Next, instead of proving knowledge of a witness $a \in \mathbb{Z}_N$ for $\mathbb{C}_{k-1}$-VPwAI, i.e. the secret key underlying the SPK, as done in [BCP21] and requiring the public key to be well-formed, the prover proves a witness for MT-GAIP, i.e. that it knows a secret $s \in \mathbb{Z}_N$ such that $E_j = [s]E_i$

Setup: Given E_0, sample a public (super)exceptional set $\mathbb{C}_{k-1} = \{c_1 = 1, c_2, \cdots, c_{k-1}\}$; Sample $a \leftarrow \mathbb{Z}_N$ and for $i = 1, \ldots, k-1$ set $E_i = [c_i a]E_0$; Output $(\mathbb{C}_{k-1}, E_0, E_1, \ldots, E_{k-1})$.

Prover: Given $(a, (\mathbb{C}_{k-1}, E_0, \cdots, E_{k-1}))$ the prover samples $b \leftarrow \mathbb{Z}_N$, and sends $E_b = [b]E_0$ to the verifier.

Verifier: Given $(\mathbb{C}_{k-1}, E_0, \cdots, E_{k-1})$ and E_b the verifier samples a random challenge $d \leftarrow \{0, \ldots, k-1\}$ and sends it to the prover.

Prover: Given $(a, \mathbb{C}_{k-1}, d)$ the prover computes $r = b - c_d \cdot a \mod N$ and sends it to the verifier.

Verifier: Given $((\mathbb{C}_{k-1}, E_0, \cdots, E_{k-1}), E_b)$ and r, return $E_b \stackrel{?}{=} [r]E_d$.

Fig. 3. The Modified Identification Protocol with Structured Public Keys.

for some pair of elliptic curves appearing in the public key, as in [BKV19]. This prevents the prover from cheating, even if the public key would not actually be correctly structured. In the case where the public key would indeed be correctly structured, knowing a witness to the MT-GAIP instance also allows to extract a witness for $\mathbb{C}_{k-1}$-VPwAI. With this, we have circumvented the need to prove well-formedness of the SPK, a main shortcoming of the protocol in [BCP21]. In comparison to the protocol in [BKV19], our ID protocol has the advantage, that it only needs a single public key to be stored, independent of the size of the public key.

Interestingly, our new ID protocol is HVZK assuming that $\mathbb{C}_{k-1}$-VPwAI is hard. Fig. 3 summarizes the resulting ID protocol.

Note that similar to the ID protocol used in CSI-FiSh, the key generation can be done by the prover. Therefore a malicious prover might choose to publish a public key that is not structured. However, based on MT-GAIP, without knowing s, it would be *still hard for an adversarial prover* to prove that it knows $s \in \mathbb{Z}_N$ such that $E_j = [s]E_i$ for some pair of elliptic curves in the public key. On the other hand, relying on the $\mathbb{C}_{k-1}$-VPwAI from Definition 3.2, we know that obtaining a from an honestly generated SPK is computationally hard. As a result, there is no real reason for the prover to generate its public key maliciously. In fact, due to several efficiency reasons that we will discuss in later sections, the prover is rather incentivized to sample the public key honestly.

Theorem 3.1. *The ID protocol presented in Fig. 3 is complete, special sound with soundness error rate $\frac{1}{k}$ for the language of MT-GAIP (Definition 2.3), and HVZK for the language of $\mathbb{C}_{k-1}$-VPwAI (Definition 3.2).*

Proof. The proof is analogous to the security proofs of the ID protocols discussed in [BKV19] and [BCP21]. However, similar to the ID protocol proposed in [BCP21], we additionally rely on the hardness of $\mathbb{C}_{k-1}$-VPwAI [BCP21], but without the need for a TTP.

Completeness. The honest prover knows the secret a for the public key $\{E_i = [c_i a]E_0\}_{i=0,\ldots,k-1}$, where $\mathbb{C}_{k-1} = \{c_0 = 0, c_1 = 1, c_2, \cdots, c_{k-1}\}$ is a public *(super-)exceptional* set. The honest verifier checks if $E_b \stackrel{?}{=} [r]E_d$. For an honestly generated proof, it holds that $[r]E_d = [b - c_d a]E_d = [b - c_d a][c_d a]E_0 = [b]E_0 = E_b$.

Honest-Verifier Zero-Knowledge. We construct a simulator that acts as follows: given the honestly sampled d, it samples r randomly from $\mathbb{Z}_N$; then sets $E_b = [r]E_d$ and returns the transcript (E_b, d, r). In both the real and the simulated transcripts, r and E_b are sampled uniformly at random, yielding indistinguishable distributions. Note that relying on the $\mathbb{C}_{k-1}$-VPwAI, obtaining the secret key a from an honestly generated SPK is computationally hard.

Special Soundness. Given two valid transcripts of the Σ-protocol, an efficient extraction algorithm extracts a witness as follows: Let (E_b, d, r) and (E_b, d', r') be two acceptable transcripts of the protocol, where $d \neq d'$, consequently $r \neq r'$ (for non-zero a). From the verification equation, one can conclude that $[r]E_d = [r']E_{d'}$ and consequently $E_d = [r' - r]E_{d'}$, which allows the extractor to obtain $r' - r$ as a solution to the MT-GAIP.

□

Remark 3.1. Note that the ID protocol used in CSI-Fish [BKV19] is special sound and HVZK for the language of MT-GAIP, and under a trusted key generation the ID protocol proposed in [BCP21] is special sound and HVZK for the language of $\mathbb{C}_{k-1}$-VPwAI. The issue with the former is that it requires $k-1$ independent secret keys, and the concern with the later is that it needs a TTP to generate the keys. Our ID protocol achieves to the best of both, as it has a *single* secret key, and *does not* require a TTP, while relying on both assumptions.

3.2 NIZK Argument and Signature Scheme

Our ID protocol from Fig. 3 can be transformed into a non-interactive zero-knowledge argument or a signature scheme in the standard ways using the Fiat–Shamir transform [FS87]. To this end, we introduce the hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{t_S \lceil \log_2 k \rceil}$, modelled as a random oracle, where t_S denotes the number of repetitions needed in the protocol. For simplicity, we will only present the resulting signature. The NIZK argument can be constructed with a similar technique, and its security can be proven in a similar way as in Lemma 3.1 in [BCP21]. We present our signature scheme in Fig. 4, and call it CSI-SharK, which stands for ***CSI**-FiSh with **Shar**ing-friendly **K**eys.*

Theorem 3.2. *Under MT-GAIP (Definition 2.3) and $\mathbb{C}_{k-1}$-VPwAI (Definition 3.2), when H is modelled as a (quantum) random oracle, then the CSI-SharK signature scheme described in Fig. 4 is sEUF-CMA secure.*

Proof. From the security of the resulting NIZK argument (similar to Lemma 3.2 in [BCP21]), we know that the modified ID protocol has special soundness and unique responses. Then, by Theorem 25 in [DFMS19], it is a quantum argument of knowledge. Moreover, since the modified protocol has λ bits of min-entropy, from Theorem 22 of [DFMS19], this shows that the CSI-SharK obtained via Fiat–Shamir is sEUF-CMA in the QROM. □

$\mathsf{KeyGen}(1^n)$**:** Given E_0, the secret key and public are generated as follows:

1. Sample $a \leftarrow \mathbb{Z}_N$;
2. Generate a public (super)exceptional set $\mathbb{C}_{k-1} = \{c_1 = 1, c_2, \cdots, c_{k-1}\}$;
3. Given E_0, for each $c_i \in \mathbb{C}_{k-1}$ set $E_i = [c_i a]E_0$.
4. Return $\mathsf{sk} = a$, $\mathsf{pk} = (E_0, E_1, \ldots, E_{k-1})$.

$\mathsf{Sign}(\mathsf{sk}, m)$**:** To sign a message m, the signer performs

1. For $i = 1, 2, \ldots, t_S$: sample $b_i \leftarrow \mathbb{Z}_N$, and set $E_{b_i} = [b_i]E_0$.
2. Set $(d_1, \ldots, d_{t_S}) = \mathsf{H}(E_{b_1}, \ldots, E_{b_{t_S}} \parallel m)$.
3. For $i = 1, 2, \ldots, t_S$: set $r_i = b_i - c_{d_i} \cdot a \pmod N$.
4. Return $\{(r_i, d_i)\}_{i=1}^{t_S}$.

$\mathsf{Verify}(\{(r_i, d_i)\}_{i=1}^{t}, m, \mathsf{pk})$**:** To verify a signature $\{(r_i, d_i)\}_{i=1}^{t_S}$ on m, one performs:

1. For $i = 1, 2, \ldots, t_S$: compute $E'_{b_i} = [r_i]E_{d_i}$.
2. $(d'_1, d'_2, \ldots, d'_{t_S}) = \mathsf{H}(E'_{b_1}, \ldots, E'_{b_{t_S}} \| m)$.
3. If $(d_1, d_2, \ldots, d_{t_S}) = (d'_1, d'_2, \cdots, d'_{t_S})$ then return valid, else output $\mathsf{invalid}$.

Fig. 4. CSI-Shark Signature Scheme.

Optimizations and Efficiency. Our basis protocol has a soundness error rate $\frac{1}{k}$. By choosing the unique base curve E_0 we can again increase this to a soundness error rate of $\frac{1}{2k-1}$ by also using the twists. This implies that we need $t_S = \lceil \mathsf{sec} \log_{2k-1} 2 \rceil$ protocol repetitions to reach a desired target soundness error of $2^{-\mathsf{sec}}$. To guarantee that the protocol is still HVZK and the SPK does not reveal any information about the secret key, we need to restrict $\mathbb{C}_{k-1}$ to superexceptional sets. As a result, the runtimes of our protocols are exactly the same as the respective versions from [BKV19] or [BCP21]. The same holds for the public-key and proof or signature sizes.

3.3 Proof of Commitments and Well-formedness of SPK

In our actively secure distributed protocols, to guarantee that the parties follow the protocol, they are asked to commit to their secret shares and prove knowledge of the committed value. Furthermore, the parties will be required to prove that they indeed act with their committed secret value on some given elliptic curve to prove the correctness of generating/updating the SPK. More precisely, each party will need to prove knowledge of a witness s to the following language, which we define for arbitrary j and a public (super)exceptional set $\mathbb{C}_j$ with integer elements $\{c_1 = 1, c_2, \cdots, c_j\}$.

$$\mathbb{L}_j := \left\{ \begin{array}{c} \Big((F_0, F'_0, E_1, E'_1, \ldots, E_j, E'_j, \mathbb{C}_j := \{c_1 = 1, c_2, \cdots, c_j\}),\ s \Big) : \\ (F'_0 = [s]F_0) \bigwedge \Big(\bigwedge_{i=1}^{j} (E'_i = [c_i s]E_i) \Big) \end{array} \right\}. \quad (1)$$

In other words, since we set $c_1 = 1$, then the prover would need to prove in a zero-knowledge manner that it knows a unique witness s for

- an instance of the GAIP, when $j = 0$.
- two simultaneous instances of the GAIP, when $j = 1$.

$\mathsf{ZK}.P_1((F_0, F'_0, E_1, E'_1, \ldots, E_j, E'_j), \{c_1 = 1, c_2, \cdots, c_j\})$**:** The prover does:

1. $b \leftarrow \mathbb{Z}_N$; Set $\hat{F}_0 \leftarrow [b]F_0$.
2. For $i = 1, \ldots, j$ compute $\hat{E}_i \leftarrow [c_i b]E_i$. Output $(\hat{F}_0, \hat{E}_1, \ldots, \hat{E}_j)$.

$\mathsf{ZK}.V_1(F_0, F'_0, \hat{F}_0, E_1, E'_1, \hat{E}_1, \ldots, E_j, E'_j, \hat{E}_j)$**:** The verifier acts as below:

1. If $E_i \neq E_0$ for any $i \in \{1, \ldots, j\}$ then sample $d \leftarrow \{0, 1\}$ and output it.
2. Else sample $d \leftarrow \{-1, 0, 1\}$ and output it.

$\mathsf{ZK}.P_2((F_0, F'_0, \hat{F}_0, E_1, E'_1, \hat{E}_1, \ldots, E_j, E'_j, \hat{E}_j), d, s)$**:** Given d, the prover computes $r \leftarrow b - d \cdot s \bmod N$, and outputs r.

$\mathsf{ZK}.V_2((F_0, F'_0, \hat{F}_0, E_1, E'_1, \hat{E}_1, \ldots, E_j, E'_j, \hat{E}_j), \{c_1 = 1, c_2, \cdots c_j\}, d, r)$**:** The verifier returns 1 if all the following checks pass, and otherwise returns 0.

1. If $d = -1$ return $\left([r]F_0'^t = \hat{F}_0\right) \wedge \bigwedge_{i=1}^{j} \left([c_i r]E_i'^t = \hat{E}_i\right)$.
2. If $d = 0$ return $\left([r]F_0 = \hat{F}_0\right) \wedge \bigwedge_{i=1}^{j} \left([c_i r]E_i = \hat{E}_i\right)$.
3. If $d = 1$ return $\left([r]F'_0 = \hat{F}_0\right) \wedge \bigwedge_{i=1}^{j} \left([c_i r]E'_i = \hat{E}_i\right)$.

Fig. 5. The HVZK Argument for Proving the Commitment and the Well-formedness of Structured Public Keys

- an instance for conjunction of the GAIP and $\mathbb{C}_{k-1}$-VPwAI, when $j = k - 1$.

The first item can be proven using the basic ID protocol from CSI-FiSh [BKV19], while for the next two items we use the techniques of conjunctive Σ-protocols. We present the underlying Σ-protocol in Fig. 5, which is obtained by the conjunction of the basic ID protocol with the *well-formedness* proof for structured public keys, presented in Section 5.1 of [BCP21]. The resulting Σ-protocol can be considered as an extension of the Σ-protocol presented in Figure 7 of [CS20], to work with *structured public keys*. We also note that since in a structured public key $c_1 = 1$, these two proofs coincide in the cases $j = 0$ and $j = 1$. Similar to [CS20], we consider two variants of the presented Σ-protocol, one when $F_0 = E_1 = \ldots = E_j = E_0$ which we call the Special case, and the other when this condition does not hold, which is called the General case. Next, we prove the security of the presented Σ-protocol.

Theorem 3.3. *The interactive argument in Fig. 5 is correct, has soundness error rate $\frac{1}{2}$ in the* General *case and soundness error rate $\frac{1}{3}$ in the* Special *case, and is computational HVZK for the language $\mathbb{L}_j$ assuming GAIP and $\mathbb{C}_j$-VPwAI.*

Proof. The proof is given in in Appendix B.1. □

Soundness Error Rate. The above theorem showed that the basic interactive argument has soundness error rate 1/2 in the General case and 1/3 in the Special case. Therefore, to achieve a target soundness error rate $2^{-\mathsf{sec}}$ for a given security parameter sec, we need to repeat the protocol sec (resp. $\lceil \mathsf{sec} \log_3 2 \rceil$) times.

Making the Protocol Non-Interactive. The Σ-protocol in Fig. 5 is an HVZK public coin interactive argument and can be turned into a NIZK argument in the standard manner using a hash function $G : \{0,1\}^* \rightarrow \{0,1\}^{\mathsf{sec}}$

in the General case, or $G : \{0,1\}^* \to \{-1,0,1\}^{\lceil \mathsf{sec} \log_3 2 \rceil}$ in the Special case. Using a 'slow' hash function for G, as in the case of CSI-FiSh, which is 2^h times slower than a normal hash function, we can reduce the number of repetitions to $t_{\mathsf{ZK}}^{\mathsf{General}} = \mathsf{sec} - h$ or $t_{\mathsf{ZK}}^{\mathsf{Special}} = \lceil (\mathsf{sec} - h) \log_3 2 \rceil$, respectively.[4] In the resulting NIZK argument, we denote the prover and verifier by $\mathsf{NIZK}.P$ and $\mathsf{NIZK}.V$.

Both the prover and the verifier need to compute a total of $(j+1)t_{\mathsf{ZK}}$ group actions throughout this protocol, ignoring the cost of the other operations, as they are negligible in comparison to group action computations. The output size of the proof is composed of the hash output and the responses. The former has a size of approximately $t_{\mathsf{ZK}}^{\mathsf{General}}$ bits ($\log_2 3^{t_{\mathsf{ZK}}^{\mathsf{Special}}} \approx t_{\mathsf{ZK}}^{\mathsf{General}}$), while the latter consists of t_{ZK} elements from $\mathbb{Z}_N$, depending on either the Special or the General case. It is interesting to note that the proof size does not depend on j.

Lemma 3.1. *The two algorithms* $\mathsf{NIZK}.P$ *and* $\mathsf{NIZK}.V$ *constitute a non-interactive zero-knowledge quantum proof of knowledge in the quantum random oracle model.*

Proof. Since the group action is free [Cou06, CLM+18], our schemes have superlogarithmic collision entropy as defined in [Unr17] and unique responses as defined in [DFMS19]. Using the results from Theorem 3.3, [Unr17] implies ZK against quantum attackers while [DFMS19], along with a challenge space superpolynomial in sec implies that our protocol is a quantum proof of knowledge. □

4 Key Generation based on CSI-FiSh

In this section, we review the current isogeny-based key generation protocols for generating the public key of CSI-FiSh. In particular, we look at the key generation of two threshold signature schemes presented in [DM20, CS20], and a Distributed Key Generation (DKG) protocol presented in [BDPV21]. We also discuss their extension to larger public keys in the same manner as is needed to extend e.g. CSI-FiSh [BKV19].

We also show that the actively secure threshold signature scheme Sashimi proposed by Cozzo and Smart [CS20], in its general form for Replicated Secret Sharing (RSS), has a security flaw in the key generation. We present a sample attack that ends up giving an honest party a wrong share after the key generation step, without the party realizing. As such, the honest party is rendered unable to sign, resulting in incorrect signatures, even if the signing protocol is correctly executed. We will therefore only focus on the full-threshold version of Sashimi, which is immune to the described attack.

At the end, we also discuss some optimizations to the DKG protocols, which allow parties to stagger the computations so as to minimize the idle time in the distributed protocol and thus optimize the overall runtime. This approach was briefly mentioned in [DM20] and we show that it can also be used in the actively

[4] As an example, we can choose $h = 16$ for $\mathsf{sec} = 128$ as is done in [BCP21] and [BKV19]. This gives $t_{\mathsf{ZK}}^{\mathsf{General}} = 112$ for the General case and $t_{\mathsf{ZK}}^{\mathsf{Special}} = 71$ for the Special case.

KeyGen: Given E_0, a TTP acts as follows:

1. For $i = 1, \ldots, k-1$, sample secrets $s_i \leftarrow \mathbb{Z}_N$ and use Shamir secret sharing to split the shares s_i into subshares $s_{i,j}$ for $j = 1, \ldots, n$.
2. Distribute $s_{1,j}, \ldots, s_{k-1,j}$ privately to party P_j.
3. Output the public key $\mathsf{pk} := (E_0, \ldots, E_{k-1})$, where $E_i = [s_i]E_0$.

Fig. 6. The key generation protocol of De Feo and Meyer's TSS [DM20].

secure case by staggering the zero-knowledge proofs needed for active security. We end each subsection by giving the optimal runtime for n parties by using this type of public key extension and also give an estimate of the communication costs between the parties. The sequential runtimes are simply expressed in terms of group actions. Regarding the communication costs, we note that elliptic curves over finite fields in the CSIDH setting can be expressed with a single parameter of size $\log p$. As $N = \#Cl(\mathcal{O}) \approx \sqrt{p}$, elements in $\mathbb{Z}_N$ can be expressed with approximately $\frac{1}{2}\log p$ bits. Finally, we choose $\mathsf{sec} = \frac{1}{4}\log p$.[5]

4.1 Key Generation of a Passively Secure TSS

De Feo and Meyer [DM20] presented the first TSS based on isogenies using Shamir secret sharing. Their TSS uses CSI-FiSh to generate distributed signatures and is proven to be secure against passive adversaries. The key generation step is done by a TTP called the *dealer*, who generates a secret key and splits it into shares using Shamir secret sharing (SSS). These shares are distributed securely to the parties and the public key is also computed by the dealer. In Figure 6, we present this protocol in the case where we have to generate $k-1$ secret elements as for the case of an extended public key. At the end of this protocol, each party P_j will hold a share $s_{i,j}$ of each secret key s_i. We also present the description of the signing protocol in Appendix A.3.

Cost. It is clear that the TTP has to generate $k-1$ secrets and distribute a total of $n(k-1)$ subshares to the different players. The TTP is left with creating the public key through the computation of $k-1$ group actions.

4.2 Full-threshold Sashimi

The original Sashimi protocol. While the TSS from the last subsection achieves passive security and requires a TTP to perform the key generation, Sashimi [CS20] aims to achieve active security and uses a Pseudo-Random Secret Sharing (PRSS) to generate a replicated secret sharing. Fig. 8 describes the algorithms underlying Sashimi, as presented in [CS20], using Fig. 7 as a subroutine. During the key generation the parties first run an instance of $\mathcal{F}_{\mathsf{Rand}}.\mathsf{PRSS}()$

[5] Choosing the security parameter this high is meant to reflect the classical security of CSIDH-based protocols against meet-in-the-middle attacks, cf. [CLM+18] and sources therein for more details.

Input: The fixed elliptic curve E_0, a set of parties Q, a secret shared element $s \in \mathbb{Z}_N$ 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 $j = 1, \ldots, t$
   - $E_{P_j} \leftarrow [s_{P_j}]E_0$; $\pi^1_{P_j} \leftarrow \mathsf{NIZK}.P((E_0, E_{P_j}), s_{P_j})$.
   - 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 $\mathsf{NIZK}.V((E_0, E_{P_i}), \pi^1_{P_i})$ and aborts if the verification algorithm fails.
5. $E^0 \leftarrow E_0$.
6. For $j = 1, \ldots, t$ do
   - Party P_j computes $E^j \leftarrow [s_{P_j}]E^{j-1}$
   - $\pi^2_{P_j} \leftarrow \mathsf{NIZK}.P((E_0, E_{P_j}, E^{j-1}, E^j), s_{P_j})$.
   - Broadcast $(E^j, \pi^2_{P_j})$ to all players.
   - All players execute $\mathsf{NIZK}.V((E_0, E_{P_j}, E^{j-1}, E^j), \pi^2_{P_j})$ and abort if the verification algorithm fails.
7. Return E^t.

Fig. 7. Group Action Computation for a Full Threshold Secret Sharing [CS20].

and form a secret sharing $\langle a_i \rangle$ of each secret key a_i (we refer the reader to the original paper for more details on $\mathcal{F}_{\mathsf{Rand}}.\mathsf{PRSS}()$ and the protocol that implements it). For RSS, each party holds multiple shares and the same share belongs to multiple parties (except for the special case of full-threshold where each party holds exactly one share of the secret). Then the parties agree on a qualified set Q and turn the RSS $\langle a_i \rangle$ into a full-threshold secret sharing over Q by properly re-arranging the shares and adding them together. This means that each party $P \in Q$ holds a single share $a_{i,P}$ for each key, which is the sum of some (possibly all) the previous shares. The formal way to pass from a replicated to full-threshold is explained in the original paper (Section 2.2 of [CS20]).

After the sharing phase the parties engage in the $\mathsf{GrpAction}$ protocol for generating the public key elements $E_1, \ldots, E_{k-1}$. Within $\mathsf{GrpAction}$, each party in Q first commits to its shares $a_{i,P}$ and attaches a proof of knowledge. Then it commits to this data using a RO-based commitment scheme and sends the commitment to the other parties. After the successful verification of the proofs, the parties start a round-robin protocol and compute the public keys E_i by using their committed secret shares. Again, the parties give a proof to ensure that they are updating the public key using the same value they committed to earlier. The NIZK proofs inside $\mathsf{GrpAction}$ protocol are given for the language

$$\mathbb{L}_j := \left\{ \left((F_0, F_0', E_1, E_1', \ldots, E_j, E_j'),\ s \right) : (F_0' = [s]F_0) \bigwedge \left(\bigwedge_{i=1}^{j} (E_i' = [s]E_i) \right) \right\},$$

KeyGen: To generate a distributed key we execute:

1. Call $\mathcal{F}_{\mathsf{Rand}}.\mathsf{Init}()$.
2. For $i \in [1, \ldots, k-1]$ 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 $\langle a_1 \rangle, \ldots, \langle a_{k-1} \rangle$ and $E_1, \ldots, E_{k-1}$.

$\mathsf{Sign}(m, \langle s \rangle)$**:** 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 $b_{i,j} \leftarrow \mathbb{Z}_N$, 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 $\{(r_i, c_i)\}_{i=1}^{t_{\mathsf{S}}}$.

Fig. 8. The Distributed Key Generation and Signing Protocols of Sashimi [CS20].

for the cases that $j = 0$ and $j = 1$. For these two cases, this language exactly coincides with the language introduced in Equation (1). As a result, the protocol presented in Fig. 5 also coincides with the one proposed in [CS20], when $j = 0, 1$.

A Security Flaw in Sashimi. Unfortunately, the ZK proofs inside $\mathsf{GrpAction}$ are not enough to guarantee that the parties follow the protocol. A malicious party can deviate from the key generation protocol in such a way, that another (honest) party might not be able to sign anymore and such that after the key generation, the produced signature will not pass the final verification, even if all parties behave honestly in the signing protocol.

The issue arises from the fact that in the key generation phase, after obtaining the secret shares $\langle a_i \rangle$ from the PRSS protocol, each party in Q has to commit to $a_{i,P}$ which is the sum of some of its replicated shares. In particular it might not commit to the single shares they got from the PRSS. Malicious parties could simply commit to a different secret value than their original secret share and generate a NIZK proof and pass the verification, thus effectively changing the "correct" share. Note that this problem only regards the adversarial shares that are also held by an honest party. The full-threshold case, where each party only holds a single share of the secret, is not affected by this issue. As a possible countermeasure, in the public key generation the parties in Q need to commit to all the shares they got from the PRSS and consistency checks need to be done by *all* the parties, not just those in Q.

A Sample Attack. Suppose that we have a $(3, 2)$-threshold access structure done with a RSS scheme. Then, we have a secret x defined as $x = x_1 + x_2 + x_3$, such that P_1 obtains $\{x_2, x_3\}$, P_2 obtains $\{x_1, x_3\}$, and P_3 gets $\{x_1, x_2\}$. Assume wlog

that P_1 is the corrupt party and that the qualified set in the key generation step is $Q = \{P_1, P_2\}$. Suppose that the parties agree on re-arranging the shares so that in the $\mathsf{GrpAction}$ protocol P_1 enters $x_{P_1} := x_2 + x_3$ and P_2 enters $x_{P_2} := x_1$. If P_1 now enters an arbitrary value y instead of x_{P_1}, then this would not be detected. This is because P_1 only commits to y and not to x_2 and x_3 individually, therefore there is no way for P_2 and P_3 to check this. As a result, the final secret key would be $x' = x_1 + y$ rather than $x = x_1 + x_2 + x_3$, so the final public key would be $E'_i = [x_1 + y]E_0$ instead of $E_i = [x_1 + x_2 + x_3]E_0$. As a result, in the threshold signing protocol, even if both parties of a qualified set $\{P_2, P_3\}$ behave honestly and follow the protocol, the resulting signature cannot be succesfully verified under the public key E'_i, as it is signed with the secret key $x = x_1 + x_2 + x_3$.

In current design of the protocol, this attack cannot be detected by the other parties, because there is no check inside the $\mathsf{GrpAction}$ protocol to ensure that the parties commit to the same secret shares sampled in the secret-sharing phase.

An Actively Secure Full-TSS from Sashimi. Here we present a slightly modified, simpler version of Sashimi for the special case of full-threshold access structure. The signing protocol of the full-threshold version is same as original scheme (shown in Fig. 8), while the full-threshold key generation protocol is described in Fig. 9. The new key generation protocol is a *full-threshold k-MT-GAIP generation protocol*, since it can generally be used to sample a k length MT-GAIP instance in a full-threshold manner. The security proof of the full-threshold version follows in Sashimi [CS20] but is specialized to the full-threshold case. For completeness we give the proof in Appendix B.2. Next, we discuss the efficiency of the resulting full-threshold DKG protocol.

Cost. We express the protocol cost in terms of Group Actions (GAs) and consider the other computational costs as negligible in comparison.

In step 4, all parties compute $k-1$ GAs, $k-1$ times execute the ZK argument in Fig. 5 with $j = 0$, and then verify each other's proofs in step 5. This can be done by all parties in parallel and results in a total cost of $(k-1)(1 + nt_{\mathsf{ZK}}^{\mathsf{Special}})$ GAs, since we are in the $\mathsf{Special}$ case. In step 7, each party first computes $k-1$ elliptic curves, runs $k-1$ times the ZK argument in Fig. 5 with $j = 1$, and then all other players can verify this proof in parallel. Because of the round-robin structure, this is repeated sequentially for all players, i.e. n times. Note that since $(E_1^0, E_i^1) = (E_0, E_{i,P_j})$, P_1 does not need to compute anything in this step. The full sequential cost of each party's round in step 7 (except for P_1) amounts to $2(k-1)t_{\mathsf{ZK}}^{\mathsf{General}}$ GA for the proof, and another $2(k-1)t_{\mathsf{ZK}}^{\mathsf{General}}$ GA for the verification, which can be done by all other players in parallel. We end up with a total naive cost of $(k-1)(n + nt_{\mathsf{ZK}}^{\mathsf{Special}} + 4(n-1)t_{\mathsf{ZK}}^{\mathsf{General}})$ GA for the full protocol. The runtime of step 7 can however be improved by using an idea similar to the staggering approach mentioned in [DM20], but where the players stagger the NIZK proofs and verifications. The idea is that at step j, party P_j computes its $k-1$ elliptic curves and then builds $\pi_{1,j}$ (while the other players are idle). As soon as this proof is finished, this proof is broadcast and P_j starts

Input: The fixed elliptic curve E_0 and a set Q of n parties.
Output: $([s_1]E_0, \ldots, [s_{k-1}]E_0)$

1. Parties individually sample $k-1$ secrets $s_i \in \mathbb{Z}_N$ shared between the parties, where $P_j \in Q$ holds $s_{1,j}, \ldots, s_{k-1,j}$ such that $s_i = \sum_{P_j \in Q} s_{i,j}$.
2. Define an ordering the players in $Q = \{P_1, \ldots, P_n\}$.
3. Each party P_j initialises an instance of $\mathcal{F}_{\mathsf{Commit}}$; call it $\mathcal{F}_{\mathsf{Commit}}^{P_j}$.
4. For $i = 1, \ldots, k-1$, each party P_j executes
   - $E_{i,P_j} \leftarrow [s_{i,j}]E_0$.
   - $\pi^1_{i,j} \leftarrow \mathsf{NIZK}.P((E_0, E_{i,P_j}), s_{i,j})$. `(Run the argument in Figure 5 for j = 0)`
   - Use $\mathcal{F}_{\mathsf{Commit}}^{P_j}$ where P_j submits input $(\mathsf{Commit}, \mathsf{id}_{P_j}, (E_{i,P_j}, \pi^1_{i,j}))$ and all other parties input $(\mathsf{Commit}, \mathsf{id}_{P_j}, \perp)$
5. For $i = 1, \ldots, k-1$
   - 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.
   - All other players execute $\mathsf{NIZK}.V((E_0, E_{i,P_j}), \pi^1_{i,j})$ and abort if the verification algorithm fails.
6. $E^0_1 \leftarrow E_0, E^0_2 \leftarrow E_0, \cdots, E^0_{k-1} \leftarrow E_0$.
7. For $j = 1, \ldots, n$
   - Party P_j computes $E^j_1 \leftarrow [s_{1,j}]E^{j-1}_1, \cdots, E^j_{k-1} \leftarrow [s_{k-1,j}]E^{j-1}_{k-1}$.
   - For $i = 1, \ldots, k-1$, compute $\pi^2_{i,j} \leftarrow \mathsf{NIZK}.P((E_0, E_{i,P_j}, E^{j-1}_i, E^j_i), s_{i,j})$. `(Run the argument in Figure 5 for j = 1)`
   - Broadcast $(E^j_1, E^j_2, \cdots, E^j_{k-1}, \pi^2_{1,j}, \ldots, \pi^2_{k-1,j})$ to all players.
   - For $i = 1, \ldots, k-1$, all other players execute $\mathsf{NIZK}.V(E_0, E_{i,P_j}, E^{j-1}_i, E^j_i)$ and abort if the verification algorithm fails.
8. Return $(E^n_1, E^n_2, \ldots, E^n_{k-1}) = ([s_1]E_0, [s_2]E_0, \cdots, [s_{k-1}]E_0)$.

Fig. 9. Full-threshold k-MT-GAIP distributed key generation protocol.

computing $\pi_{2,j}$. At the same time, all other players verify $\pi_{1,j}$, which takes the same amount of computational effort as building the proof. This continues until all players have finally verified $\pi_{k-1,j}$ in the last step (where P_j is idle). It is easy to see, that the round of P_j has a total sequential cost $k - 1 + 2kt^{\mathsf{General}}_{\mathsf{ZK}}$ GA. We finally end up with a total sequential cost of

$$n(k-1)(1 + t^{\mathsf{Special}}_{\mathsf{ZK}}) + (n-1)2kt^{\mathsf{General}}_{\mathsf{ZK}} \tag{2}$$

for the entire protocol.

For the communication cost, we see that both in step 4 and 7, each party publishes $k-1$ elliptic curves and proofs. We can bound the total communication output per player by $(k-1)\,(10 + \log p)\,\frac{1}{4}\log p$.[6]

[6] In order get this compact formula, we are ignoring the parameter h of the slow hash function, as well as the gain resulting from the $\mathsf{Special}$ case. As such, this formula represents an upper bound of the total communication per player.

4.3 CSI-RAShi: A Distributed Key Generation Protocol

In [BDPV21], Beullens et al. proposed a DKG protocol based on Shamir's secret sharing [Sha79] called CSI-RAShi, that allows a set of parties to generate a public key $E_1 = [a]E_0$ in a distributed manner, with a being their shared secret. Since Pedersen commitments [Ped92] do not exist as such in the isogeny setting, they can not be used for parties to verify the correctness of their shares. As a way out, the authors introduce *Piecewise Verifiable Proofs* (PVP), which allow parties to prove and verify correctness of their collective and individual shares using ZK proofs in a much faster way than the naive approach. PVPs are a major contribution of [BDPV21], however we skip their definition here, as our later improvements are mainly related to CSI-RAShi's bottleneck, which are standard ZK proofs needed in the public-key computation step. We also refer to the original source for the details about the complaint-disqualification mechanisms between the players. In Figure 10, we present CSI-RAShi extended to generating $k-1$ keys. We also call it *Shamir k-MT-GAIP generation protocol*, as it can be used to sample a k length MT-GAIP instance based on Shamir secret sharing. We note that by simply using one key, we recover the original version from [BDPV21].

Cost. The runtime of CSI-RAShi is very similar to the DKG of Sashimi. A major difference is that in the VSS step, the parties first compute $k-1$ elliptic curves $R_i^{(j)}$ and build their proofs using PVPs. The cost of the PVP is dominated by the main proof (e.g. $\tilde{\pi}_i^{(j)}$) which costs $t_{\mathsf{ZK}}^{\mathsf{General}}$. Adding $n-1$ verifications for each such proof, the VSS ends up costing $(k-1)(2+nt_{\mathsf{ZK}}^{\mathsf{General}})$ sequential group actions. Steps 4 and 5 amount to the same cost as step 6 of the Sashimi group action, i.e. $2(k-1)t_{\mathsf{ZK}}^{\mathsf{General}}$ per proof and again per verification. The difference with the protocol in Fig. 9 is that party P_1 also has to run these proofs. Finally, we end up with a total naive cost of $(k-1)(2+n+5nt_{\mathsf{ZK}}^{\mathsf{General}})$.[7] By again staggering the proofs and verifications, as described in the Sashimi DKG, we can further reduce the sequential cost of steps 4 and 5. Optimally, we end up with a total sequential cost of

$$(n+2)(k-1) + nt_{\mathsf{ZK}}^{\mathsf{General}}(3k-1)\,. \tag{3}$$

For the communication cost, in the first step, each party publishes $2(k-1)$ elliptic curves and $k-1$ main proofs of the PVPs, then sends $k-1$ shares and proof pieces to each of the $n-1$ other players. In step 4, each party further pub-

[7] In [BDPV21], another optimization is discussed which reduces the dominant term from $5nt_{\mathsf{ZK}}^{\mathsf{General}}$ to $4nt_{\mathsf{ZK}}^{\mathsf{General}}$. This is achievable by parties already starting to create their own proofs in their idle time. This could also be used in Sashimi, but only has a minor effect when combined with staggering. We choose to omit it here for simplicity, given that staggering will reduce the dominant term to $3nt_{\mathsf{ZK}}^{\mathsf{General}}$ in either case.

Input: An elliptic curve E_0, a set $\{P_1, \ldots, P_n\}$ of n parties.
Output: A public key $([a_1]E_0, \ldots, [a_{k-1}]E_0)$

Verifiable Secret Sharing:

1. For $i = 1, \ldots, k-1$, each player P_j samples a degree $t-1$ polynomial $f_i^{(j)}(x)$ with coefficients from $\mathbb{Z}_N$, as well as a uniformly random elliptic curve $R_i^{(j)}$ and computes $R_i'^{(j)} = [f_i^{(j)}(0)]R_i^{(j)}$. Then, each player constructs a PVP
$$\pi_i^{(j)} = (\tilde{\pi}_i^{(j)}, \pi_{i,1}^{(j)}, \ldots, \pi_{i,n}^{(j)})$$
which includes a main proof $\tilde{\pi}_i^{(i)}$ as well as individual proof pieces $\pi_{i,l}^{(j)}$ for each other player P_l. Finally, each player publishes the main part $((R_i^{(j)}, R_i'^{(j)}), \tilde{\pi}_i^{(j)})$ and sends $(f_i^{(j)}(l), \pi_{i,l}^{(j)})$ privately to P_l. The main proof $\tilde{\pi}_i^{(j)}$ allows to verify the statement $R_i'^{(j)} = [f_i^{(j)}(0)]R_i^{(j)}$, while a proof piece $\pi_j^{(i)}$ allows a player to verify correctness of their share $f_i^{(j)}(l)$.
2. For $i = 1, \ldots, k-1$, each player P_j verifies all the proofs $\tilde{\pi}_i^{(l)}$ and $\pi_{i,j}^{(l)}$ of all other players P_l with respect to their statements. Whenever a proof fails, players broadcast complaints and the interaction of the concerned players are scrutinized by the other players. This might result in the disqualification of players that didn't follow the protocol properly (see [BDPV21] for more details).
3. In the end, all the honest players agree on the same set of qualified players $\mathcal{Q} \subset \{1, \ldots, n\}$. At this point the joint secret keys are implicitly defined as $a_i = \sum_{j \in \mathcal{Q}} f_i^{(j)}(0)$. Each party P_j derives their share of a_i as $a_{i,j} = \sum_{l \in \mathcal{Q}} f_i^{(l)}(j)$.

Computing the Public Key:
Set $F_1^0 \leftarrow E_0, \ldots, F_{k-1}^0 \leftarrow E_0$.

4. For $j = 1, \ldots, n$, in a round-robin way, P_j computes $F_1^j \leftarrow [f^{(j)}(0)]F_1^{j-1}, \ldots, F_{k-1}^j \leftarrow [f^{(j)}(0)]F_{k-1}^{j-1}$, then builds the proofs
$$\pi_i'^{(j)} \leftarrow \mathsf{NIZK}.P(R_i^{(j)}, R_i'^{(j)}, F_{j-1}^i, F_j^i),$$
by running the argument in Figure 5 for $j = 1$.
5. These proofs are verified by all parties. Whenever a proof fails, parties again scrutinize the interaction and can disqualify malicious players. In the honest majority setting, parties can even reconstruct missing information if needed, so that the public key implicitly defined by point 3.
6. In the end, the parties return their public key
$$(F_1^{|\mathcal{Q}|}, F_2^{|\mathcal{Q}|}, \ldots, F_{k-1}^{|\mathcal{Q}|}) = ([a_1]E_0, [a_2]E_0, \ldots, [a_{k-1}]E_0)\,.$$

Fig. 10. The DKG Protocol CSI-RAShi [BDPV21] for an Extended Public Key.

lishes $k-1$ curves and proofs. After some arithmetic,[8] the total communication output per player can be expressed as $\frac{1}{4}\log p(k-1)(8n+17+\log p)$.

[8] Using the description in [BDPV21], the communication content in a PVP is composed of a main proof of $\mathsf{sec}(4(n+1)+\log N)$ bits and $2(n+1)$ proof shares, each of sec bits, resulting in a total of $\mathsf{sec}(6(n+1)+\log N)$.

5 Key Generation based on CSI-SharK

Next, we revisit the key generation protocols based on CSI-FiSh (given in Sec. 4) and show, that by using SPKs (as used in CSI-SharK) instead of standard extended public keys (as used in CSI-FiSh), the cost of these DKG protocols is reduced to a lower cost than current optimal cases (discussed in Sec. 4). This gain mainly arises from the fact, that the VSS steps need to be performed for a single key only, and during the DKG the $k-1$ independent NIZK proofs can be replaced with a single proof of Fig. 5 with $j = k-1$. We then discuss optimizations to the sequential cost of the protocol, which allows us to even further reduce the cost of these protocols by optimally splitting these larger proofs into chunks and staggering them. We note that this approach works when using (multiples of) the same secret key in these protocols, which happens in the SPK case.

5.1 Structured Key Generation in a Passively Secure TSS

Fig. 11 presents a passively secure DKG protocol, based on the protocol from [DM20], which uses the SPKs. For the associated signing protocol we refer to Appendix A.3.

Theorem 5.1. *Under the Power-DDHA and $\mathbb{C}_{k-1}$-VPwAI assumptions, the protocol in Fig. 11 is correct and simulatable.*

Proof. The proof is completely analogous to the proof of Theorem 1 in [DM20]. □

Computational cost. We can see that the TTP computes pk using $k-1$ group actions. While this is the same number of computations as in Sec. 4.1, the TTP in this case only has to generate a single secret and distribute n shares of these secrets, reducing the communication by a factor $k-1$.

5.2 Structured Sashimi

In Fig. 12, we present a new variant of the full-threshold DKG protocol presented in Fig. 9, that can be used to generate a SPK. Similar to the previous case, as this protocol can be used to sample a $\mathbb{C}_k$-VPwAI instance in a full-threshold fashion, we call it *full-threshold $\mathbb{C}_k$-VPwAI generation protocol*. To achieve active security, parties compute NIZK proofs for the language from equation (1) for the cases $j = 0$ and $j = k-1$.

KeyGen: Given E_0, a TTP acts as follows:

1. Sample a secret $s \leftarrow \mathbb{Z}_N$ and define a (super)exceptional set $\mathbb{C}_{k-1} = \{c_1 = 1, c_2, \ldots, c_{k-1}\}$.
2. Use SSS to split s into subshares s_j for $j = 1, \ldots, n$ and distribute s_j privately to party P_j.
3. Output the public key $\mathsf{pk} := (E_0, \ldots, E_{k-1})$, where $E_i = [c_i s]E_0$.

Fig. 11. Passive distributed key generation protocol with structured public key.

Input: The fixed elliptic curve E_0 and a set Q of n parties.
Output: $[s]E_0, [c_2 s]E_0, \cdots, [c_{k-1}s]E_0$

1. Parties agree on a super-exceptional set $\mathbb{C}_{k-1} = \{c_1 = 1, c_2, \ldots, c_{k-1}\}$.
2. Parties individually sample a secret $s_j \in \mathbb{Z}_N$, such that $s = \sum_{P_j \in Q} s_j$.
3. Define an ordering the players in $Q = \{P_1, \ldots, P_n\}$.
4. Each party P_j initialises an instance of $\mathcal{F}_{\mathsf{Commit}}$; call it $\mathcal{F}_{\mathsf{Commit}}^{P_j}$.
5. Each party P_j computes
   - $E_{P_j} \leftarrow [s_j]E_0$.
   - $\pi_j^1 \leftarrow \mathsf{NIZK}.P((E_0, E_{P_j}), s_j)$. (Run the argument in Fig. 5 for $j = 0$)

   All parties call $\mathcal{F}_{\mathsf{Commit}}^{P_j}$ where P_j submits input $(\mathsf{Commit}, \mathsf{id}_{P_j}, (E_{P_j}, \pi_j^1))$ and all other parties input $(\mathsf{Commit}, \mathsf{id}_{P_j}, \perp)$
6. For $j = 1, \ldots, n$
   - 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 $P_i \neq P_j$, party P_i executes $\mathsf{NIZK}.V((E_0, E_{P_j}), \pi_j^1)$ and aborts if the verification algorithm fails.
7. $E_1^0 \leftarrow E_0, E_2^0 \leftarrow E_0, \cdots, E_{k-1}^0 \leftarrow E_0$.
8. For $j = 1, \ldots, n$ do
   - Party P_j computes

$$E_1^j \leftarrow [s_j]E_1^{j-1}, E_2^j \leftarrow [c_2 s_j]E_2^{j-1}, \cdots, E_{k-1}^j \leftarrow [c_{k-1}s_j]E_{k-1}^{j-1},$$
$$\pi_j^2 \leftarrow \mathsf{NIZK}.P((E_0, E_{P_j}, E_1^{j-1}, E_1^j, \cdots, E_{k-1}^{j-1}, E_{k-1}^j), \mathbb{C}_{k-1}, s_j).$$

   - Broadcast $(E_1^j, E_2^j, \cdots, E_{k-1}^j, \pi_j^2)$ to all players.
   - All players execute $\mathsf{NIZK}.V((E_0, E_{P_j}, E_1^{j-1}, E_1^j, \ldots, E_{k-1}^{j-1}, E_{k-1}^j), \mathbb{C}_{k-1}, \pi_j^2)$ and abort if the verification algorithm fails.
9. Return $([s]E_0, [c_2 s]E_0, \cdots, [c_{k-1}s]E_0) = (E_1^n, E_2^n, \cdots, E_{k-1}^n)$.

Fig. 12. Full-threshold $\mathbb{C}_k$-VPwAI Generation Protocol.

In Appendix A.4, we also introduce a full-threshold signature scheme based on this DKG protocol, which is a variant of the signing algorithm in Sashimi. We further prove the security of the DKG and the signature scheme against active adversaries in Appendix B.3.

Computational cost. We can see that steps 5 and 6 of Fig. 12 have a total cost of $1 + nt_{\mathsf{ZK}}^{\mathsf{Special}}$, as all steps can be done in parallel by all players. Step 8 consists of each player computing $k-1$ isogenies and then building a NIZK proof for $j = k-1$, which is then verified in parallel by all other players. Each such step therefore costs $(k-1)(1 + 2t_{\mathsf{ZK}}^{\mathsf{General}})$. The exception is player P_1, which has already computed $E_{P_j} = E_1^1$ and therefore can exclude it from the NIZK proof. Furthermore, the proofs and verification of P_1's step are in the $\mathsf{Special}$ case, so we end up with the total cost of

$$(n-2)t_{\mathsf{ZK}}^{\mathsf{Special}} + (k-1)(n + 2t_{\mathsf{ZK}}^{\mathsf{Special}} + (n-1)2t_{\mathsf{ZK}}^{\mathsf{General}}). \tag{4}$$

We note that the dominant term in this equation scales with $2nkt_{\mathsf{ZK}}^{\mathsf{General}}$, which is already lower than the cost of the protocol reviewed in Sec. 4.2.

We can however further optimize step 8 for $k > 2$. Currently, each player computes the full proof with $j = k - 1$, while other players wait, then this proof is verified while the prover waits. Instead, we can subdivide the one full proof of $k - 1$ elements into r smaller proofs of $j = \lceil \frac{k-1}{r} \rceil$ elements. The idea is to *stagger* these smaller proofs and verifications steps as was done inSec. 4.2. While the proving party computes the first proof, the other players are idle. After finishing this proof, it is published, and the proving player computes its second proof while the other players verify the first one. This is repeated r times. In the last round, the other parties verify the rth proof, while the prover is idle. As such, there is a big overlap between the computations of the prover and the verifiers, reducing the overall idle time. Assuming for simplicity that r divides $k-1$, it is clear that each of the proofs should contain the pair (E_0, E_{P_j}) as the reference curves, as well as $\frac{k-1}{r}$ other curves, which should be proven to have been computed correctly. Given the staggered approach, we end up with a total of $r + 1$ proof-verification cycles, where each costs $\left(\frac{k-1}{r} + 1\right) t_{\mathsf{ZK}}$ group actions. By assuming the prover has already precomputed the (E_0, E_{P_j}) part of its first proof, as explained above, we can reduce the cost of the first cycle to $\frac{k-1}{r} t_{\mathsf{ZK}}$ group actions. Including the costs of the elliptic curve computations, this yields a total of $k - 1 + \frac{k-1}{r} t_{\mathsf{ZK}} + r\left(\frac{k-1}{r} + 1\right) t_{\mathsf{ZK}}$ sequential group actions per round. It is clear that this cost is minimized for $r = \sqrt{k-1}$. We end up with a cost per round-robin step of $k - 1 + \left((\sqrt{k-1} + 1)^2 - 1\right) t_{\mathsf{ZK}}$ for players $P_2, \ldots, P_n$.

We are left with establishing the cost of P_1's round. Again, P_1 only needs to compute $k - 2$ curves and consequently prove correctness of these $k - 2$ elements. Using the same process as before we end up with an optimal $r' = \sqrt{k-2}$, i.e. we have a total of $\sqrt{k-2}+1$ cycles with proofs/verifications of cost $(\sqrt{k-2}+1)t_{\mathsf{ZK}}^{\mathsf{Special}}$ per cycle. Adding the cost of step 5, we end up with the total sequential cost of the DKG protocol of:[9]

$$\begin{aligned} T^{DKG}(n, k, \mathsf{sec}) = n(k-1) + \left(n + (\sqrt{k-2} + 1)^2\right) t_{\mathsf{ZK}}^{\mathsf{Special}} \\ + (n-1)\left((\sqrt{k-1} + 1)^2 - 1\right) t_{\mathsf{ZK}}^{\mathsf{General}}. \end{aligned} \tag{5}$$

The cost in equation (4) is dominated by the term $2knt_{\mathsf{ZK}}^{\mathsf{General}}$, while here, this term is reduced to $n(k+2\sqrt{k})t_{\mathsf{ZK}}^{\mathsf{General}}$. For large k, this gives another improvement by almost a factor of 2.

It is easy to see that our protocol also strongly reduces the communication cost needed between the parties. First of all, by using only one secret instead of $k - 1$, we reduce the communication cost of the VSS step by a factor $k - 1$. Similarly, in the public-key computation step, we reduce the number of proofs from $k - 1$ to $\sqrt{k-1}$ per player, and even to 1 in the non-optimized case. Only the number of elliptic curves published in the public-key computation step stays constant, when compared to the non-structured case. Expressed in bits, we find $\frac{1}{4} \log p \,(4k + 7n + 7 + \log p)$ in the case with one single proof and

$\frac{1}{4}\log p\left(4k+7n+6+\frac{1}{2}\log p+(\sqrt{k-1}+1)(1+\frac{1}{2}\log p)\right)$ in the optimized case, as the communication output per player. It is easy to see that asymptotically for large k, both these cases yield a gain factor of $\frac{1}{4}(10+\log p)$, when compared to the protocol in Sec. 4.2.

5.3 Structured CSI-RAShi

We conclude this section by presenting a CSI-RAShi-based DKG protocol for an SPK in Fig. 13. The protocol is called *Shamir $\mathbb{C}_k$-VPwAI generation protocol*, as it can also be used to sample a $\mathbb{C}_k$-VPwAI instance in a fully distributed manner using Shamir secret sharing. We also present an actively secure TSS based this DKG in Appendix A.5.

Theorem 5.2. *The protocol of Fig. 13 satisfies the consistency requirement, assuming GAIP and $\mathbb{C}_{k-1}$-VPwAI and satisfies the secrecy requirement, further assuming the $\mathbb{C}_{k-1}$-Decisional GAIP.*

Proof. The proof is given in Appendix B.4. □

Computational cost. In contrast to the extension with multiple secret keys, the SPK extension in Fig. 13 requires only *a single execution* of the VSS step, instead of $k-1$ repetitions, and so it is independent of the public key size. The cost of this step is $2+nt_{\mathsf{ZK}}^{\mathsf{General}}$ group actions. Furthermore, parties do not have to repeat separate ZK proofs multiple times, but can rather compute one big proof for all $k-1$ elements. This results in a cost per round-robin step of $(k-1)(1+2t_{\mathsf{ZK}}^{\mathsf{General}})$, resulting in the total sequential cost of

$$2+nt_{\mathsf{ZK}}^{\mathsf{General}}+n(k-1)(1+2t_{\mathsf{ZK}}^{\mathsf{General}}) \tag{6}$$

for the entire protocol. It is easy to see that the dominant term scales with $2nkt_{\mathsf{ZK}}^{\mathsf{General}}$, which is lower than the cost of the protocol in Sec. 4.3. We can improve this cost by using the staggering approach described in Sec. 5.2, in which we split the main proof into $\sqrt{k-1}$ smaller proofs of size approximately $\sqrt{k-1}$. A difference to the cost established in that section is again that P_1 is not in the Special case and has to compute $k-1$ curves (as opposed to $k-2$ in Fig. 12). We end up with the final cost

$$2+n(k-1)+t_{\mathsf{ZK}}^{\mathsf{General}}\left(n(\sqrt{k-1}+1)^2+1\right). \tag{7}$$

The dominating term in CSI-RAShi scales as $3nkt_{\mathsf{ZK}}^{\mathsf{General}}$, while using structured keys can reduce this to $n(k+2\sqrt{k})t_{\mathsf{ZK}}^{\mathsf{General}}$. For large k, this gives an improvement of almost a factor 3.

[9] Note that equation (5) is slightly simplified, since in general, the square roots within this equation are not integers. If e.g. $\sqrt{k-1}$ is not an integer, we construct $r=\lceil\sqrt{k-1}\rceil$ proofs, each of size at most $\lceil\frac{x}{r}\rceil$. This means that we can substitute the terms of type $(\sqrt{x}+1)^2$ by the term $(\lceil\sqrt{x}\rceil+1)(\lceil\frac{x}{\lceil\sqrt{x}\rceil}\rceil+1)$ (where e.g. $x\in\{k-1,k-2\}$) in order to upper bound the actual cost.

Input: An elliptic curve E_0, a set $\{P_1, \ldots, P_n\}$ of n parties.
Output: A public key $([a]E_0, [c_2a]E_0, \cdots, [c_{k-1}a]E_0)$
Verifiable Secret Sharing:

1. Each player P_i, for $i = 1, \ldots, n$, samples a degree $t-1$ polynomial $f^{(i)}(x)$ with coefficients from $\mathbb{Z}_N$, as well as a uniformly random elliptic curve $R^{(i)}$ and computes $R'^{(i)} = [f^{(i)}(0)]R^{(i)}$. Then, each player constructs a PVP
$$\pi^{(i)} = (\tilde{\pi}^{(i)}, \pi_1^{(i)}, \ldots, \pi_n^{(i)})$$
which includes a main proof $\tilde{\pi}^{(i)}$ as well as individual proof pieces $\pi_j^{(i)}$ for each other player P_j. Finally, each player publishes the main part $((R^{(i)}, R'^{(i)}), \tilde{\pi}^{(i)})$ and sends $(f^{(i)}(j), \pi_j^{(i)})$ privately to P_j. The main proof $\tilde{\pi}^{(i)}$ allows verifying the statement $R'^{(i)} = [f^{(i)}(0)]R^{(i)}$, while a proof piece $\pi_j^{(i)}$ allows a player to verify the correctness of their share $f^{(i)}(j)$.
2. Now, each player P_j verifies all the proofs $\tilde{\pi}^{(i)}$ and $\pi_j^{(i)}$ of all other players with respect to their statements. Whenever a proof fails, players broadcast complaints and the interaction of the concerned players are scrutinized by the other players. This might result in the disqualification of players that didn't follow the protocol.
3. In the end, all the honest players agree on the same set of qualified players $\mathcal{Q} \subset \{1, \ldots, n\}$. At this point the joint secret key is implicitly defined as $a = \sum_{i \in \mathcal{Q}} f^{(i)}(0)$. Each party P_j derives their share of a as $a_j = \sum_{i \in \mathcal{Q}} f^{(i)}(j)$.

Computing the *Structured* Public Key:

4. Parties agree on a super-exceptional set $\mathbb{C}_{k-1} = \{c_1 = 1, c_2, \ldots, c_{k-1}\}$.
5. In a round-robin way, the qualified players compute
$$F_i^1 \leftarrow [f^{(i)}(0)]F_{i-1}^1, F_i^2 \leftarrow [c_2 f^{(i)}(0)]F_{i-1}^2, \ldots, F_i^{k-1} \leftarrow [c_{k-1} f^{(i)}(0)]F_{i-1}^{k-1},$$
where $F_0^1 = \cdots = F_0^{k-1} = E_0$. At each step, player P_i publishes the proof
$$\pi'^{(i)} \leftarrow \mathsf{NIZK}.P((R^{(i)}, R'^{(i)}, F_{i-1}^1, F_i^1, \ldots, F_{i-1}^{k-1}, F_i^{k-1}, \mathbb{C}_{k-1}), f^{(i)}(0)),$$
by running the NIZK argument in Fig. 5 for $j = k-1$.
6. This proof is verified by all parties. Whenever a proof fails, parties again scrutinize the interaction and can disqualify malicious players. In the honest majority setting, parties can even reconstruct missing information if needed, so that the public key is implicitly defined by point 3.
7. In the end, the parties return their structured public key
$$(F_{|\mathcal{Q}|}^1, F_{|\mathcal{Q}|}^2, \ldots, F_{|\mathcal{Q}|}^{k-1}) = ([a]E_0, [c_2a]E_0, \ldots, [c_{k-1}a]E_0)\,.$$

Fig. 13. Shamir $\mathbb{C}_k$-VPwAI Generation Protocol.

Again, we note that our scheme reduces the communication cost in the VSS by a factor $k-1$, as only a single PVP is needed. Similarly, in the public-key computation step, we reduce the number of proofs from $k-1$ to $\sqrt{k-1}$ per player, or 1 in the non-optimized case, while the number of published curves stays $k-1$. For the case with a single proof, we find the total cost of

$\frac{1}{4}\log p\,(4k + 6n + 10 + \log p)$, while in the optimized case, we find $\frac{1}{4}\log p(4k + 6n + 9 + (\sqrt{k-1}+1)(1+\frac{1}{2}\log p))$, as the communication output per player.

For asymptotically large k, in comparison to the non-structured case, we get a gain factor of the communication of $\frac{1}{4}(8n + 17 + \log p)$, when compared to Sec. 4.3. We note that the gain increases with the number of parties. This is due to the fact that the size of the PVPs depends on the number of players.

6 Parallel Executions and Performances

Next, we discuss the benefits of parties having multiple CPU threads at their disposal when executing isogeny-based signatures and NIZK proofs. Given the fact that multi-core CPUs are standard, we want to consider their impact on the schemes presented in this work. As observed in all threshold protocols, to compute a curve $[x]E_0$, where x is shared among multiple parties, having to adopt a sequential round-robin communication structure between the parties seems to be an unavoidable fact. However, we observe that in the cases that we need to compute more than one curve, e.g. $[x_1]E_0, \ldots, [x_{k-1}]E_0$, be it either as a single party (as in the CSI-FiSh or CSI-SharK) or by several parties (as in the threshold variants of them or DKG protocols), these computations are in general independent of each other and can therefore very easily be parallelized.

In the case of *non-threshold* protocols, where only one party runs the algorithm, the parallelization can be done using different threads (or cores) of a CPU. Namely, each thread of the CPU can compute a subset of these curves, and will finish the total computation considerably faster than the consecutive approach described in the original CSI-FiSh [BKV19] and its follow-up schemes [CS20,BDPV21]. As an example, assuming a party has C independently accessible threads, they can compute a CSI-SharK (or CSI-FiSh) signature for the consecutive cost of $\lceil t_S/C \rceil$ group actions instead of the full t_S, because of the independence of the commitments. In Table 3 of Appendix A.7, we show the impact of parallelizing signature schemes in this way. We observe that using 8 cores with a public key of size $k = 2$ (64 B) already allows for signing and verifying as fast as using a public key of size $k = 2^{12} + 1$ (256 KB) with a single core.

In the case of *threshold* protocols, the same idea can be applied to NIZK arguments, such as the one in Fig. 5, reducing the proof cost from $(j+1)t_{\mathsf{ZK}}$ to $(j+1)\lceil t_{\mathsf{ZK}}/C \rceil$ group actions. Since all commitments can be independently computed, and the j individual commitments are parallelizable, we can further reduce this to $\lceil (j+1)t_{\mathsf{ZK}}/C \rceil$ group actions, effectively reducing the sequential costs of our algorithms by a factor C, up to some constant terms. Furthermore, while parties are still forced to adopt a sequential structure due to the round robins, they can however run multiple such round-robins in parallel to fill up the idle time. This was already observed in [DM20] for passively secure threshold protocols. Here we extend this idea to actively secure threshold case, where parties also need to generate ZK proofs and verify the proofs of other parties.

In Table 2, we present estimates of the performances of our actively secure threshold signature schemes based on CSI-SharK for different parameter sets. We compare our schemes with the DKGs from [CS20, BDPV21] and the signature scheme from [CS20]. For these estimates, we use the formulas established throughout this work, including the results for signatures from Appendix A.6. We compare them to the original time complexity formulas from the relevant sources. We use the (conservative) approximate cost of 40 ms per group action that can be expected when combining the benchmarks presented in [BKV19] with the optimizations from [MR18] on a 3.5 GHz processor [CLM+18]. The table indicates that SPKs give a strong performance benefit, even when compared to the most optimal case without structured keys.

Since multiple threads are standard in modern CPU architecture, our results show that isogeny-based signature and threshold schemes become quite practical by virtue of the parallelizability of their computations. For instance, 3 parties, each with 16 cores, could sign in 28 seconds by using a 16 KB large public key, while 8 parties would need 94 seconds with the same parameters. Key generation in these settings can be done in a few minutes and verification will take 40 ms.

7 Conclusion

We presented CSI-SharK as a new variant of CSI-FiSh [BKV19], that allows one to build more efficient distributed protocols than those based on CSI-FiSh. CSI-SharK is based on a modified version of the Σ-protocol underlying CSI-FiSh [BKV19]. A *key* difference between CSI-SharK and CSI-FiSh is that CSI-SharK uses SPKs, as recently introduced in [BCP21]. At the cost of an additional computational assumption, CSI-SharK improves CSI-FiSh in a number of ways. Public keys with $k-1$ elliptic curves are now generated using a single secret, instead of $k-1$, which means that only one secret needs to be generated and stored, independent of the public key size. This is most noticeable in distributed key generation protocols, where secrets are further split into parts and distributed among the parties. The heavy zero-knowledge proofs in current state-of-the-art protocols can be strongly reduced in numbers and even merged, when working with structured public keys. This reduces even the most optimal implementations by a factor 3 and the communication cost by a much greater factor, e.g. up to 130 for Sashimi and about $132 + \frac{3}{4}n$ for CSI-RAShi, where n is the number of parties.

While these results make threshold schemes in the isogeny setting much more practical, we further presented a general strategy for parallel computations, exploiting the independence of commitments in zero-knowledge proofs and signature schemes. We show that by parallelizing computations, isogeny-based thresh-

[9] We note that by using the optimizations from Appendix A.6, the runtime of our signature is about twice as fast as the naive approach presented in [CS20]. This is independent of the public key structure and is readily applicable to signature schemes without structured public keys as well, such as [CS20].

Table 2. Comparison of computational and communication cost for different instances of the full-threshold Sashimi DKG (upper table) and CSI-RAShi (lower table) for the CSIDH-512 parameter set. We compare both schemes in the CSI-FiSh (extended public key) and CSI-SharK cases (structured public key) for different public key sizes $k-1$. We compare *naive* implementations (as presented in their original paper or by simply using structured public keys) and optimized (*staggered*) implementations, as discussed in Sec. 4 and Sec. 5. We also indicate the communication cost (output per party) for each of these cases. We note that the communication costs in the CSI-FiSh case do not change when using the staggering optimization, while it does so in the CSI-SharK setting. In the full-threshold case, we further indicate the signature cost, as established in Appendix A.6. For completeness, we also compare secret key sizes $|\mathsf{sk}|$ per party in the CSI-FiSh and CSI-SharK cases in the lower table. Runtimes are estimated based on the conservative estimate that a group action computation takes 40 ms. For better readability, some communication cost and secret key size entries are omitted in the tables, as they are simply the same results as in the block above.

Full-Threshold Case	$k-1$	CSI-FiSh naive [CS20]	CSI-FiSh staggered (Sec. 4)	CSI-FiSh comm.	CSI-SharK naive (Sec. 5)		CSI-SharK staggered (Sec. 5)		Signing App. A.6
3 parties	2^4	17 min	10 min	131 kB	7.3 min	9.1 kB	5.5 min	21 kB	12 min
each with	2^8	4.5 h	2.6 h	2.0 MB	1.9 h	24 kB	65 min	84 kB	7.2 min
1 core	2^{12}	73 h	42 h	33 MB	31 h	264 kB	16 h	517 kB	4.5 min
3 parties	2^4	67 s	38 s		27 s		21 s		47 s
each with	2^8	18 min	10 min		7 min		4 min		28 s
16 cores	2^{12}	4.8 h	2.6 h		1.9 h		60 min		18 s
8 parties	2^4	3.1 min	1.8 min		80 s		60 s		147 s
each with	2^8	50 min	28 min		21 min		12 min		94 s
16 cores	2^{12}	13 h	7.4 h		5.6 h		2.9 h		87 s

Shamir secret sharing	$k-1$	CSI-FiSh naive [BDPV21]	CSI-FiSh staggered (Sec. 4)	CSI-FiSh comm.	$\lvert\mathsf{sk}\rvert$	CSI-SharK naive (Sec. 5)		CSI-SharK staggered (Sec. 5)		$\lvert\mathsf{sk}\rvert$
3 parties	2^4	18 min	13 min	138 kB	512 B	8.5 min	9.5 kB	6.5 min	26 kB	32 B
each with	2^8	4.7 h	3.3 h	2.1 MB	8 kB	2.2 h	25 kB	65 min	89 kB	32 B
1 core	2^{12}	76 h	53 h	34 MB	128 kB	35 h	265 kB	18 h	522 kB	32 B
3 parties	2^4	69 s	48 s			32 s		24 s		
each with	2^8	18 min	13 min			8.2 min		4.7 min		
16 cores	2^{12}	4.9 h	3.3 h			2.2 h		68 min		
8 parties	2^4	2.9 min	2.1 min	148 kB		85 s	10 kB	65 s	26 kB	
each with	2^8	47 min	33 min	2.3 MB		22 min	25 kB	12 min	89 kB	
16 cores	2^{12}	12 h	8.8 h	36 MB		5.8 h	265 kB	3.0 h	522 kB	

old (and non-threshold) signatures become truly practical and competitive in the post-quantum realm.

As an independent contribution, we revealed a flaw in the DKG protocol of Sashimi, which can allow an honest party to end up with a wrong share after the protocol, thus preventing it from generating correct signatures, even after a correctly executed signing protocol.

We think that the very design of the CSI-SharK public keys, its Σ-protocol and some of our proposed structured DKG protocols can offer many advantages for different applications and hope that the structure may be further exploited to design more practical isogeny-based cryptographic primitives.

Acknowledgments

We would like to thank Prof. Nigel Smart for his valuable comments. This work has been supported in part by the Defense Advanced Research Projects Agency (DARPA) under contract No. HR001120C0085, by the FWO under an Odysseus project GOH9718N, by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101020788 - Adv-ERC-ISOCRYPT), by CyberSecurity Research Flanders with reference number VR20192203, by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program under project PICOCRYPT (grant agreement No. 101001283), by the Spanish Government under project PRODIGY (TED2021-132464B-I00), and by the Madrid Regional Government under project BLOQUES (S2018/TCS-4339). The last two projects are co-funded by European Union EIE, and Next Generation EU/PRTR funds.

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, DARPA, the US Government, the Spanish Government, Cyber Security Research Flanders or the FWO. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

References

BCP21. Karim Baghery, Daniele Cozzo, and Robi Pedersen. An isogeny-based ID protocol using structured public keys. In M.B. Paterson, editor, *Cryptography and Coding - 18th IMA International Conference, IMACC 2021, Oxford, UK, December 14-15, 2021, Proceedings*, volume 13129 of *Lecture Notes in Computer Science*, pages 179–197. Springer, 2021.

BCPS18. Anurag Bishnoi, Pete L Clark, Aditya Potukuchi, and John R Schmitt. On zeros of a polynomial in a finite grid. *Combinatorics, Probability and Computing*, 27(3):310–333, 2018.

BDPV21. Ward Beullens, Lucas Disson, Robi Pedersen, and Frederik Vercauteren. CSI-RAShi: Distributed key generation for CSIDH. In Jung Hee Cheon and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021, Daejeon, South Korea, July 20-22, 2021, Proceedings*, volume 12841 of *Lecture Notes in Computer Science*, pages 257–276. Springer, 2021.

BKV19. Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. CSI-FiSh: Efficient isogeny based signatures through class group computations. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 227–247, Kobe, Japan, December 8–12, 2019. Springer, Heidelberg, Germany.

BS20. Xavier Bonnetain and André Schrottenloher. Quantum security analysis of CSIDH. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part II*, volume 12106 of *Lecture*

Notes in Computer Science, pages 493–522, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany.

CD22. Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH (preliminary version). 2022. https://ia.cr/2022/975.

CJS14. Andrew Childs, David Jao, and Vladimir Soukharev. Constructing elliptic curve isogenies in quantum subexponential time. *Journal of Mathematical Cryptology*, 8(1):1–29, 2014.

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, *Advances in Cryptology – ASIACRYPT 2018, Part III*, volume 11274 of *Lecture Notes in Computer Science*, pages 395–427, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany.

Cou06. Jean Marc Couveignes. Hard homogeneous spaces. *IACR Cryptol. ePrint Arch.*, 2006:291, 2006.

CS20. Daniele Cozzo and Nigel P. Smart. Sashimi: Cutting up CSI-FiSh secret keys to produce an actively secure distributed signing protocol. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 169–186, Paris, France, April 15–17, 2020. Springer, Heidelberg, Germany.

CSCDJRH20. Jorge Chávez-Saab, Jesús-Javier Chi-Domínguez, Samuel Jaques, and Francisco Rodríguez-Henríquez. The sqale of CSIDH: Square-root Vélu quantum-resistant isogeny action with low exponents. Technical report, Cryptology ePrint Archive, Report 2020/1520, 2020., 2020.

DF17. Luca De Feo. Mathematics of isogeny based cryptography. *arXiv preprint arXiv:1711.04062*, 2017.

DFMS19. Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Security of the Fiat-Shamir transformation in the quantum random-oracle model. In *Annual International Cryptology Conference*, pages 356–383. Springer, 2019.

DG19. Luca De Feo and Steven D. Galbraith. SeaSign: Compact isogeny signatures from class group actions. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 759–789, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany.

DHK+23. Julien Duman, Dominik Hartmann, Eike Kiltz, Sabrina Kunzweiler, Jonas Lehmann, and Doreen Riepel. Generic models for group actions. Cryptology ePrint Archive, Report 2023/186, 2023. https://eprint.iacr.org/2023/186.

DKL+20. Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. SQISign: Compact post-quantum signatures from quaternions and isogenies. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part I*, volume 12491 of *Lecture Notes in Computer Science*, pages 64–93, Daejeon, South Korea, December 7–11, 2020. Springer, Heidelberg, Germany.

DLSV20. Anders Dalskov, Eysa Lee, and Eduardo Soria-Vazquez. Circuit amortization friendly encodings and their application to statistically secure multiparty computation. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 213–243. Springer, 2020.

DM20. Luca De Feo and Michael Meyer. Threshold schemes from isogeny assumptions. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 12111 of *Lecture Notes in Computer Science*, pages 187–212, Edinburgh, UK, May 4–7, 2020. Springer, Heidelberg, Germany.

EKP20. Ali El Kaafarani, Shuichi Katsumata, and Federico Pintore. Lossy CSI-FiSh: Efficient signature scheme with tight reduction to decisional CSIDH-512. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 12111 of *Lecture Notes in Computer Science*, pages 157–186, Edinburgh, UK, May 4–7, 2020. Springer, Heidelberg, Germany.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany.

GJKR07. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, January 2007.

GPS17. Steven D Galbraith, Christophe Petit, and Javier Silva. Identification protocols and signature schemes based on supersingular isogeny problems. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 3–33. Springer, 2017.

JD11. David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In Bo-Yin Yang, editor, *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*, pages 19–34, Tapei, Taiwan, November 29 – December 2 2011. Springer, Heidelberg, Germany.

MM22. Luciano Maino and Chloe Martindale. An attack on SIDH with arbitrary starting curve. 2022. https://ia.cr/2022/1026.

MR18. Michael Meyer and Steffen Reith. A faster way to the CSIDH. In Debrup Chakraborty and Tetsu Iwata, editors, *Progress in Cryptology - INDOCRYPT 2018: 19th International Conference in Cryptology in India*, volume 11356 of *Lecture Notes in Computer Science*, pages 137–152, New Delhi, India, December 9–12, 2018. Springer, Heidelberg, Germany.

Ped92. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140, Santa Barbara, CA, USA, August 11–15, 1992. Springer, Heidelberg, Germany.

Pei20. Chris Peikert. He gives C-sieves on the CSIDH. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 463–492, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany.

Rob22. Damien Robert. Breaking SIDH in polynomial time. 2022. https://ia.cr/2022/1038.

RS06. Alexander Rostovtsev and Anton Stolbunov. Public-key cryptosystem based on isogenies. *IACR Cryptol. ePrint Arch.*, 2006:145, 2006.

SD21. Surbhi Shaw and Ratna Dutta. Identification scheme and forward-secure signature in identity-based setting from isogenies. In Qiong Huang and Yu Yu, editors, *Provable and Practical Security - 15th International Conference, ProvSec 2021, Guangzhou, China, November 5-8, 2021, Proceedings*, volume 13059 of *Lecture Notes in Computer Science*, pages 309–326. Springer, 2021.

Sha79. Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

Sho94. Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

Sie35. Carl Siegel. Über die classenzahl quadratischer zahlkörper. *Acta Arithmetica*, 1(1):83–86, 1935.

Sil09. Joseph H Silverman. *The arithmetic of elliptic curves*, volume 106. Springer Science & Business Media, 2009.

Sto10. Anton Stolbunov. Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves. *Advances in Mathematics of Communications*, 4(2):215, 2010.

Sto12. Anton Stolbunov. Cryptographic schemes based on isogenies. 2012.

Unr17. Dominique Unruh. Post-quantum security of Fiat-Shamir. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 65–95. Springer, 2017.

YAJ+17. Youngho Yoo, Reza Azarderakhsh, Amir Jalali, David Jao, and Vladimir Soukharev. A post-quantum digital signature scheme based on supersingular isogenies. In *International Conference on Financial Cryptography and Data Security*, pages 163–181. Springer, 2017.

A Signatures

In this section, we discuss standard and distributed signatures in both the CSI-FiSh and CSI-SharK setting. We start by first introducing sigma and ID protocols, then move on to introducing signature schemes and distributed signature schemes in the next subsection. Later, we present signature schemes related to the DKG schemes with structured public keys from Section 5. We then discuss execution strategies for distributed signature schemes in order to minimize the idle time of the players. Finally, in the last subsection, we discuss the influence of using multiple cores on the non-distributed signature schemes CSI-FiSh and CSI-SharK.

Throughout this section, we assume the structured public keys have $k-1$ elements, meaning that the signature process has a soundness error rate of $\frac{1}{2k-1}$, by also using the twists. To achieve a soundness error of $2^{-\mathsf{sec}}$, we therefore have to repeat the signing process $t_S = \lceil \mathsf{sec} \log_{2k-1} 2 \rceil$ times. We define the hash function $H : \{0,1\}^* \to \{0,1\}^{t_S \lceil \log_2 2k-1 \rceil}$. Furthermore, for the elements $c_i \in \mathbb{C}_k$, we define $c_{-i} = -c_i$.

A.1 Sigma Protocols and ID protocols

Next, we recall the definition of sigma protocols (Σ-protocols) and identification schemes. Here the algorithms are Probabilistic Polynomial-Time (PPT), unless mentioned. Let λ be a security parameter and let $X = X(\lambda)$ and $W = W(\lambda)$ be sets. Let $\mathbf{R}$ be a relation on $X \times W$ that defines a language $Ł = \{x \in X : \exists \omega \in W, \mathbb{R}(x, \omega) = 1\}$. Given $x \in Ł$, an element $\omega \in W$ such that $\mathbf{R}(x, \omega) = 1$ is called a witness. Let $\mathbf{R}$ be a PPT algorithm such that $\mathcal{R}(\lambda)$ outputs pairs (x, ω) such that $\mathbf{R}(x, \omega) = 1$.

A sigma-protocol (Σ-protocol) for the relation $\mathbf{R}$ is a 3-round interactive protocol between two PPT algorithms: a prover P and a verifier V. P holds a witness ω for $x \in Ł$ and V is given x. P first sends a value a to V, and then V answers with a challenge c , and finally P answers with z. V accepts or rejects the proof. The triple $\mathsf{trans} = (a, c, z)$ is called a transcript of the Σ-protocol. A Σ-protocol is supposed to satisfy *Completeness*, *Honest Verifier Zero-Knowledge* (HVZK), and *Special Soundness* defined below.

Definition A.1 (Completeness). *A Σ-protocol Π with parties (P, V) is* complete for $\mathcal{R}$*, if for all $(x, \omega) \in \mathbf{R}$,*

$$\Pr\left[\mathsf{trans}(P(\mathbf{R}, x, \omega) \leftrightarrow V(\mathbf{R}, x)) \text{ is accepted by } V\right] = 1 \ ,$$

where trans *denotes the transcript of the protocol.*

Definition A.2 (HVZK). *A Σ-protocol satisfies* HVZK for $\mathcal{R}$*, if there exists a PPT algorithm $\mathcal{S}$ that given $x \in X$, can simulate the* trans *of the scheme, s.t. for all $x \in Ł$, $(x, \omega) \in \mathbf{R}$,*

$$\mathsf{trans}(P(\mathbf{R}, x, \omega) \leftrightarrow V(\mathbf{R}, x)) \approx \mathsf{trans}(\mathcal{S}(\mathbf{R}, x) \leftrightarrow V(\mathbf{R}, x))$$

where $\mathsf{trans}(P(\cdot) \leftrightarrow V(\cdot))$ indicates the transcript of Π with (P, V), and $\approx$ denotes the indistinguishability of transcripts.

Definition A.3 (Special Soundness). *The Σ-protocol Π with parties (P, V) is* special sound for $\mathcal{R}$*, if there exists a PPT extractor* Ext*, such that for any $x \in Ł$, given two valid transcripts (a, c, z) and (a, c', z') for the same message a but $c \neq c'$, then $\mathsf{Ext}(a, c, z, c', z')$ outputs a witness ω for the relation $\mathbf{R}$.*

Identification Protocols. An ID protocol is a special case of a Σ-protocol between two parties (P, V), with respect to a hard relation defined by a key generator KGen, as $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$, where one thinks of sk as a witness for the public key pk.

A.2 Signatures

Digital Signature Schemes. A digital signature scheme consists of three algorithms (KeyGen, Sign, Verify) that are defined as bellow.

- $\mathsf{KeyGen}\left(1^{\lambda}\right)$ is a PPT algorithm that given the security parameter λ, returns a public key pk and a secret key sk.
- $\mathsf{Sign}\left(\mathsf{sk}, m\right)$ is a PPT algorithm that given the secret key sk and a message m returns a signature σ.
- $\mathsf{Verify}\left(\mathsf{pk}, (\sigma, m)\right)$ is a deterministic polynomial-time algorithm that given the pk, a signature σ and a message m returns either 1 (valid) or 0 (invalid).

A primary security notion for a disgital signature scheme is defined as follows.

Definition A.4 (Strong Existential Unforgeability under Chosen Message Attacks). *A signature scheme* $\Pi = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *is said to be strong Existentially Unforgeable under adaptive Chosen-Message Attacks (sEU-CMA) if for all PPT adversaries* $\mathcal{A}$*,*

$$\left|\Pr\begin{bmatrix}(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KGen}(1^{\lambda}),\ \sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}, m_i) \text{ for } 1 \le i \le k; \\ (m,\sigma) \leftarrow \mathcal{A}^{\mathsf{Sign}(.)}(\mathsf{pk}, (m_i,\sigma_i)_{i=1}^{k}) : \mathsf{Verify}(m,\sigma,\mathsf{pk}) = 1 \wedge (m,\sigma) \notin Q\end{bmatrix}\right|,$$

is negligible in λ*, where* $Q := \{(m_1,\sigma_1)\cdots,(m_k,\sigma_k)\}$ *is the set of the messages requested by* $\mathcal{A}$ *and the signatures returned from the signing oracle.*

Threshold Signature Schemes. A threshold signature scheme allows a set of qualified parties to jointly sign a message m, and produce a signature σ on it that can be verified using a single public key pk. More precisely, a threshold signature scheme with respect to a (n,t)-threshold access structure is dedfined as follows

Definition A.5. *A* threshold 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 a set of secret keys* sk_i*, one secret key for every party*[10]
- $\mathsf{Sign}\left(\{\mathsf{sk}_i\}_{i\in Q}, m\right)$ *is a randomized signing algorithm that takes as inputs a* qualified set *of private keys and a message and returns a signature on the message.*
- $\mathsf{Verify}\left(\mathsf{pk}, (\sigma, m)\right)$ *is a deterministic verification algorithm that takes as inputs the public key and a signature* σ *on a message* m *and outputs a bit which is equal to one if and only if the signature on* m *is valid.*

Informally, security for a threshold signature scheme implies that an unqualified set of parties cannot forge a signature on a new message. In our distributed signatures, we also require that a valid output signature should be indistinguishable from the signature produced by the signing algorithm of the underlying non-thresholdized scheme with the same public key.

A.3 Passively Secure Threshold Signing Protocols

We introduce the passively secure TSS from [DM20] as well as its counterpart using structured public keys.

[10] Since the focus of this paper is on Shamir and full-threshold secret sharing, here we can limit ourselves to the case where each party gets a single share of the secret.

CSI-FiSh-based TSS. At the end of the DKG from Figure 6, each party P_j will hold a share $s_{i,j}$ of the secret key s_i, for $i = 1, \cdots, k-1$ s.t. $s_i = \sum_{j \in Q} s_{i,j} \cdot L^Q_{0,i,j}$ for a set Q. To sign a message m, any qualified set $Q = \{1, \ldots, n\}$ of n parties generates a signature as in Figure 14.

$\mathsf{Sign}(m, \langle s \rangle)$**:** Set $E_1^0 \leftarrow E_0, E_2^0 \leftarrow E_0, \cdots, E_{t_s}^0 \leftarrow E_0$, and then act as follows:

1. In a round robin for $j = 1, \ldots, n$, each player samples $b_i^j \leftarrow \mathbb{Z}_N$, then computes $E_i^j = [b_i^j] E_i^{j-1}$ for $i = 1, \cdots, t_S$.
2. Set $(d_1, \ldots, d_{t_s}) = \mathsf{H}(E_1^n, E_2^n, \ldots, E_{t_s}^n \parallel m)$.
3. For $i = 1, \cdots, t_S$, each player computes an broadcasts $r_i^j = b_i^j - \mathsf{sign}(d_i) \cdot s^j_{d_i} \cdot L^Q_{0,d_i,j} \pmod N$.
4. Return $\{(r_i, d_i)\}_{i=1}^{t_s}$, where $r_i = \sum_{j=1}^n r_i^j$.

Fig. 14. Passively Secure Signature Scheme Based on CSI-FiSh [DM20].

CSI-SharK-based TSS. At the end of the DKG from Figure 11, each party P_j will hold a *single* share s_j of the secret key s, s.t. $s = \sum_{j \in Q} s_j \cdot L^Q_{0,j}$ for a set Q. To sign a message m, any qualified set $Q = \{1, \ldots, n\}$ of n parties generates a signature as in Figure 15.

$\mathsf{Sign}(m, \langle s \rangle)$**:** Set $E_1^0 \leftarrow E_0, E_2^0 \leftarrow E_0, \cdots, E_{t_s}^0 \leftarrow E_0$, and then act as follows:

1. In a round robin for $j = 1, \ldots, n$, each player samples $b_i^j \leftarrow \mathbb{Z}_N$, then computes $E_i^j = [b_i^j] E_i^{j-1}$ for $i = 1, \cdots, t_S$.
2. Set $(d_1, \ldots, d_{t_s}) = \mathsf{H}(E_1^n, E_2^n, \ldots, E_{t_s}^n \parallel m)$.
3. For $i = 1, \cdots, t_S$, each player computes an broadcasts $r_i^j = b_i^j - \mathsf{sign}(d_i) \cdot c_{d_i} s_j \cdot L^Q_{0,j} \pmod N$.
4. Return $\{(r_i, d_i)\}_{i=1}^{t_s}$, where $r_i = \sum_{j=1}^n r_i^j$.

Fig. 15. Passively Secure Signature Scheme Based on CSI-SharK

Theorem A.1. *Under the Power-DDHA and $\mathbb{C}_{k-1}$-VPwAI, the signature protocol of Figure 15 is correct and simulatable.*

Proof. The proof is analogous to the proof of [DM20, Theorem 1]. □

A.4 Structured Sashimi with Full-Threshold Secret Sharing

At the end of the key generation from Figure 6, each party P_j will hold a Shamir share s_j of the secret key s, s.t. $s = \sum_{j \in Q} s_j \cdot L^Q_{0,j}$ for a set Q. To sign a message m, any qualified set $Q = \{1, \ldots, n\}$ of n parties can use the passively secure signing protocol in Figure 15 and generate a signature.

$\mathsf{Sign}(m, \langle a \rangle)$**:** For a set of (qualified) parties $Q = \{P_1, \ldots, P_n\}$ to sign a message m they act as follows:

1. For $i = 1, \ldots, t_{\mathsf{S}}$: Given E_0, the set of Q parties run the *full-threshold k-MT-GAIP generation protocol* in Figure 9 for $k = 2$ and generate $E'_i := [b_i]E_0$, and party P_j stores $b_{i,j}$ as a share of $\langle b_i \rangle$ [a].
2. The parties locally compute $(d_1, \ldots, d_{t_{\mathsf{S}}}) \leftarrow H(E'_1 \| \ldots \| E'_{t_{\mathsf{S}}} \| m)$.
3. For $i = 1, \ldots, t_{\mathsf{S}}$, each party P_j computes $r_{i,j} \leftarrow b_{i,j} - \mathsf{sign}(d_i) \cdot c_{d_i} a_j$.
4. The parties broadcast their values $r_{i,j}$ and locally compute $r_i \leftarrow \sum_{j=1}^{n} r_{i,j}$.
5. Output $\{(r_i, d_i)\}_{i=1}^{t_{\mathsf{S}}}$.

[a] Note that, for $k = 2$ the protocols in Figures 9 and 12 are equivalent and can be used interchangeably.

Fig. 16. Actively Secure Full-Threshold Signing Protocol with Structured Public Keys.

$\mathsf{Sign}(m, \langle a \rangle)$**:** To sign a message m, a set of (qualified) parties $Q = \{P_1, \ldots, P_n\}$ acts as follows,

1. For $i = 1, \ldots, t_{\mathsf{S}}$: Given E_0, the set of Q parties run the *full-threshold k-MT-GAIP generation protocol* in Figure 9 for $k = 2$ and generate $E'_i := [b_i]E_0$, and party P_j stores $b_{i,j}$ as a share of $\langle b_i \rangle$ [a].
2. The parties locally compute $(d_1, \ldots, d_{t_{\mathsf{S}}}) \leftarrow H(E'_1 \| \ldots \| E'_{t_{\mathsf{S}}} \| m)$.
3. For $i = 1, \ldots, t_{\mathsf{S}}$, each party P_j computes $r_{i,j} \leftarrow b_{i,j} - \mathsf{sign}(d_i) \cdot c_{|d_i|} a_j L^Q_{0,j}$.
4. The parties broadcast their values $r_{i,j}$ and locally compute $r_i \leftarrow \sum_{j=1}^{n} r_{i,j}$.
5. Output $\{(r_i, d_i)\}_{i=1}^{t_{\mathsf{S}}}$.

[a] For $k = 2$ the protocols in Figures 9 and 12 are equivalent and can be used interchangeably.

Fig. 17. Actively Secure Threshold Signing Protocol with Structured Public Keys Based on (n, t)-Shamir Secret Sharing.

Similarly, at the end of the DKG from Figure 9, each party P_j will hold a share s_j of the secret key s, s.t. $s = \sum_{j \in Q} s_j$ for a set Q. Then, using the full-threshold actively secure signing protocol in Figure 16, a set of Q parties will be able to sign a message. The proof closely follows that in Sashimi [CS20], and is adapted to the CSI-SharK case. For the sake of completeness, we give the proof in Appendix B.2.

A.5 Distributed Shamir-based Actively Secure Signature

Using our Shamir $\mathbb{C}_k$-VPwAI Generation Protocol, presented in Figure 13, a set of parties can generate the secret and structure public key of our actively secure threshold signature scheme.

At the end of the key generation, the n parties will obtain a Shamir sharing of the secret key $\langle a \rangle$. After that, in order to sign a message m, any set of qualified parties Q use the actively secure signing protocol described in Figure 17.

Security readily follows from that of the key generation in Theorem 5.2, since then the simulator has the adversarial shares of the key, which it can use to simulate the signing protocol exactly in the same way as in Theorem B.2.

A.6 Optimal Executions of Signature Schemes

Throughout this work, we have shown, how using SPKs naturally increases the efficiency of distributed key generation. Unfortunately, this advantage does not immediately transfer to signatures as there, the protocols, such as Figures 9 and 12 are executed for $k = 2$ only, so we do not have an advantage of using structured elements. In this section, we discuss how to use the strategy described in Section 6 and optimize the execution of distributed signature schemes to reduce the idle time as much as possible. These results are applicable to both structured and non-structured extended public keys with $k - 1$ elements and therefore also readily apply to the signature scheme presented in [CS20].

Since the commitment values $b_1, \ldots, b_{t_S}$ of these signatures are all independently computable, the t_S separate round-robins Figures 16 and 17 all can actually be computed in parallel, by choosing the order of the players in these round-robins as different cyclic permutations of the original set.

Assume for simplicity that we have $n = t_S$ parties. For $j = 1, \ldots, n$, the players compute one step towards the commitment and one proof, then verify the $t_S - 1$ other proofs, all in parallel. Adding the t_S invocations of steps 4 and 5 in Figures 9 and 12, we find the total cost of

$$T^{\mathsf{Sign}}_{n=t_S}(n, k, \mathsf{sec}) = (1 + nt^{\mathsf{Special}}_{\mathsf{ZK}})t_{\mathsf{S}} + (n-1)\left(1 + 2t_{\mathsf{S}}t^{\mathsf{General}}_{\mathsf{ZK}}\right) .$$

If $n > t_S$, $n - t_S$ parties will stay idle during the commitment and proofs in each step. However, we have to realize that the number of verifications per round increases to t_S instead of $t_S - 1$, since the idle parties will have to verify all proofs. Yet, players can simply postpone this verification to a later round, and verify it whenever they are idle again, waiting for other players' proofs to finish. After the very last round, each player will have one more of these verifications to complete. In the case, where $n < t_S$, we can compute at most n elements at the same time, so we have to repeat each round $\lceil \frac{t_S}{n} \rceil$ times. Note that the cost per round is $1 + 2nt^{\mathsf{General}}_{\mathsf{ZK}}$, as we have to verify at most $n - 1$ proofs. We can summarize the cost of our actively secure signatures as

$$T^{\mathsf{Sign}}(n, k, \mathsf{sec}) = (1 + nt^{\mathsf{Special}}_{\mathsf{ZK}})t_{\mathsf{S}} + \begin{cases} \lceil \frac{t_S}{n} \rceil (n-1)\left(1 + 2nt^{\mathsf{General}}_{\mathsf{ZK}}\right), & \text{if } n \leq t_S, \\ (n-1)(1 + 2t_{\mathsf{S}}t^{\mathsf{General}}_{\mathsf{ZK}}) + 2t^{\mathsf{General}}_{\mathsf{ZK}}, & \text{if } n > t_S. \end{cases} \tag{8}$$

It is interesting to note that this means choosing $n \geq t_S$ is in general favorable to $n < t_S$. In particular, this means that running the SSS-based TSS from Figure 17 can be more efficient by adding more parties in case the threshold is below t_S. These considerations lead to the results presented in Table 2.

A.7 Signature Schemes with Multiple CPU Threads

As discussed in Section 6, since the commitments needed in the signature schemes CSI-SharK and CSI-FiSh are independent, parties with multiple independently accessible CPU threads can compute these signatures for the consecutive cost of $\lceil t_S/C \rceil$ group actions instead of the full t_S. Table 3 estimates the runtimes of CSI-SharK (and CSI-FiSh) using multi-core CPUs for different sets of parameters.

We can see that by using multi-core CPUs, both CSI-SharK and CSI-FiSh become very practical. For example, using a 16-core CPU and $k-1 = 2^4$ results in a signature scheme with public key size of 1 KB, signature size of 759 B, where key generation takes 40 ms, and both signing and verification take 80 ms.

Table 3. Parameter choices and empirical performance estimation for CSI-SharK (and CSI-FiSh). #Cores: Number of cores in the CPU, k: number of curves in the public key, t_S: number of parallel executions, h: "slowness" parameter in hash function, $|x|$: size of x. Run times are estimated based on the previous results that each GA takes 40 ms.

#Cores	$k-1$	t_S	h	|sk|	|pk|	|sig|	KGen	Sign	Verify
1	2^0	71	16	16 B	64 B	2307 B	40 ms	2840 ms	2840 ms
	2^4	23	15	16 B	1 KB	759 B	640 ms	920 ms	920 ms
	2^8	13	12	16 B	16 KB	436 B	10 s	520 ms	520 ms
	2^{12}	9	11	16 B	256 KB	306 B	164 s	360 ms	360 ms
8	2^0	71	16	16 B	64 B	2307 B	40 ms	360 ms	360 ms
	2^4	23	15	16 B	1 KB	759 B	80 ms	120 ms	120 ms
	2^8	13	12	16 B	16 KB	436 B	1280 ms	80 ms	80 ms
	2^{12}	9	11	16 B	256 KB	306 B	20 s	80 ms	80 ms
16	2^0	71	16	16 B	64 B	2307 B	40 ms	200 ms	200 ms
	2^4	23	15	16 B	1 KB	759 B	40 ms	80 ms	80 ms
	2^8	13	12	16 B	16 KB	436 B	640 ms	40 ms	40 ms
	2^{12}	9	11	16 B	256 KB	306 B	10 s	40 ms	40 ms

B Security proofs

B.1 Proof of Theorem 3.3

Proof. We show the proof for the General case. The Special case is analogous.

Correctness. An honest prover knows a secret s such that $F_0' = [s]F_0$, and $E_i' = [c_i s]E_i$ for all $i = 1, \cdots, j$ and $\{c_1 = 1, c_2, \cdots, c_j\}$. If $c = 0$, then the honest verifier checks whether $[r]F_0 = \hat{F}_0$ and $[c_i r]E_i = \hat{E}_i$ for all i, and $\{c_1 = 1, c_2, \cdots, c_j\}$. Since $[r]F_0 = [b]F_0$ and $[c_i r]E_i = [c_i b]E_i$ and this is equal to $\hat{F}_0$ and $\hat{E}_i$ for all i and $\{c_1 = 1, c_2, \cdots, c_j\}$, then the verifier accepts. If $c = 1$, then honest verifier checks whether $[r]F_0' = \hat{F}_0$ and $[r]E_i' = \hat{E}_i$ for all i and $\{c_1 = 1, c_2, \cdots, c_j\}$. Since $[r]F_0' = [b-s]F_0' = [b-s]([s]F_0) = [b]F_0$ and $[c_i r]E_i' =$

$[c_i(b-s)]E'_i = [c_ib - c_is]\,([c_is]E_i) = [c_ib]E_i$ and this is equal to $\hat{E}_i$ for all i and $\{c_1 = 1, c_2, \cdots, c_j\}$, the verifier accepts. This shows that an honestly generated proof is always accepted by an honest verifier.

Special Soundness. Given two accepting transcripts $\pi = ((\hat{F}_0, \hat{E}_1, \ldots, \hat{E}_j), d, r)$ and $\pi' = ((\hat{F}_0, \hat{E}_1, \ldots, \hat{E}_j), d', r')$ of the protocol obtained by rewinding, where $d \neq d'$, and hence $r \neq r'$ (unless $s = 0$), we can construct an efficient extraction algorithm for the witness s as follows. We conclude that

$$[r]F_0 = \hat{F}_0 \ \wedge\ [c_ir]E_i = \hat{E}_i \quad \wedge \quad [r']F'_0 = \hat{F}_0 \ \wedge\ [c_ir']E'_i = \hat{E}_i,$$

for all $i \in \{1, \ldots, j\}$, and $\{c_1 = 1, c_2, \cdots, c_{j-1}\}$. This implies that we have $F'_0 = [-r']([r]F_0) = [r - r']F_0$ and $E'_i = [-c_ir']([c_ir']E'_i) = [-c_ir']([c_ir]E_i) = [c_i(r - r')]E_i$ for all i, and therefore that $c_is = c_i(r - r')$ for all i and $\{c_1 = 1, c_2, c_3, \cdots, c_j\}$. Therefore from the first equation and the fact that $c_1 = 1$, then we conclude that $s = r - r'$.

Honest Verifier Zero-Knowledge (HVZK). To prove the HVZK, we construct a simulator that given $\{c_1 = 1, c_2, \cdots, c_j\}$ and the honestly generated challenge d, which is sampled at random from $\{0, 1\}$ for the General case (from $\{-1, 0, 1\}$ in the Special case), simulates the transcript of the protocol. To this end, the simulator first samples r at random from $\mathbb{Z}_N$, then sets

- $\hat{F}_0 = [r]F_0$ and $\hat{E}_i = [c_ir]E_i$ if $d = 0$,
- $\hat{F}_0 = [r]F'_0$ and $\hat{E}_i = [c_ir]E'_i$ if $d = 1$,
- $\hat{F}_0 = [r]F'^t_0$ and $\hat{E}_i = [c_ir]E'^t_i$ if $d = -1$.

When the input to the proof is from $\mathbb{L}_j$, then the simulation is perfect. If it is not the case, then the commitments also look like they come from a uniform distribution, as they are deterministic functions of r, which is uniform. Therefore it is computationally hard for an adversary to distinguish an honestly generated proof from a simulated proof, and the argument is computationally HVZK. □

B.2 Security Proof of the Key Generation in Full-threshold Sashimi

The proof is analogous to the one in Section 4.2 of [CS20], but for the special case of full-threshold. In Figure 18, we define the functionality $\mathcal{F}_{\mathsf{DKG}}$. The functionality works with n parties according to a full-threshold access structure. The functionality takes $(n-1)\cdot(k-1)$ adversarial shares, completes them to get an additive share of random elements $s_1, \ldots, s_{k-1}$ and outputs the public key for CSI-FiSh consisting of the elements $E_1 = [s_1]E_0, \ldots, [s_{k-1}]E_0$.

Then we prove the following theorem.

Theorem B.1. *The protocol in Figure 9 securely implements the functionality $\mathcal{F}_{\mathsf{DKG}}$ (defined in Figure 18) in the $\mathcal{F}_{\mathsf{Commit}}$-hybrid model against an active adversary corrupting up to $n-1$ parties.*

The functionality works with parties $P_1, \ldots, P_n$, where P_l is honest as follows:

KeyGen: Upon receiving $(\mathbb{Z}_N, E_0, k-1, \mathsf{Init})$ from all parties:

1. The functionality receives shares $a_{i,j} \in \mathbb{Z}_N$ with $i = 1, \ldots, k-1$ and $j \in [n] \backslash l$ from the adversary.
2. The functionality samples random $s_1, \ldots, s_{k-1} \leftarrow \mathbb{Z}_N$ and then sets

$$s_{1,l} := s_1 - \sum_{i \neq l} s_{1,i}, \ldots, s_{k-1,l} := s_{k-1} - \sum_{i \neq l} s_{k-1,i}$$

   as the shares of the honest party.
3. The functionality then computes $E_1 := [s_1]E_0, \cdots, E_{k-1} := [s_{k-1}]E_0$ and sends them to the adversary and waits for an input from the adversary.
4. If the input is abort then abort execution. Otherwise send the public keys to the honest party.

Fig. 18. Distributed key generation functionality for Sashimi: $\mathcal{F}_{\mathsf{DKG}}$

KeyGen Simulation: Let P_l be the honest party. $\mathcal{A}$ and $\mathcal{S}$ engage in an execution of the *full-threshold k-MT-GAIP distributed key generation* protocol (given in Figure 9).

In the commit phase each party needs to commit with $\mathcal{F}_{\mathsf{Comm}}$ to the elliptic curves holding its secret shares of $s_1, \ldots, s_{k-1}$ and the associated proofs. The simulator, which does not have the shares of P_l, commits with $\mathcal{F}_{\mathsf{Commit}}^{P_l}$ to arbitrary inputs, instead of committing to $E_{1,P_l}, \ldots, E_{k-1,P_l}$ and the associated proofs of knowledge. When later it is asked to open this, it will simply equivocate the commitment so that it can be opened to the correct elliptic curves and proofs that it is able to compute after extracting the adversarial shares.

From the $\pi^1_{P_i}$, given in the commit phase, $\mathcal{S}$ is able to extract the values $s_{i,j}$ entered by $\mathcal{A}$ in the first round of proofs. The extracted values $s_{i,j}$ are now passed to the functionality $\mathcal{F}_{\mathsf{DKG}}$, which completes them to a valid set of shares of the secret and returns the corresponding public key $E_1 = [s_1]E_0, \ldots, E_{k-1} = [s_{k-1}]E_0$ to the simulator.

At this point, $\mathcal{S}$ has all the adversarial shares and the public key. However, the honest shares $s_{1,l}, \ldots, s_{k-1,l}$ are unknown to $\mathcal{S}$. Even though it does not have the honest shares, it can equivocate the commitment $\mathcal{F}_{\mathsf{Commit}}$ so that it can be opened to

$$E_{1,P_l} := \left[-\textstyle\sum_{i \neq l} s_{1,i}\right] E_1, \ldots, E_{k-1,P_l} := \left[-\textstyle\sum_{i \neq l} s_{k-1,i}\right] E_{k-1}$$

and the associated simulated proofs, which the simulator can compute as it has the correct statement.

The commitments can now be opened. Now $\mathcal{A}$ and $\mathcal{S}$ proceed with the round-robin protocol for computing the public keys as in step 8 of Figure 9.

When it comes the turn of party P_l, the curves it gets from party P_{l-1} are

$$E_1^{l-1} = \left[\textstyle\sum_{i=1}^{l-1} s_{1,i}\right] E_0, \ldots, E_{k-1}^{l-1} = \left[\textstyle\sum_{i=1}^{l-1} s_{k-1,i}\right] E_0$$

The simulator will then output the curves

$$E_1^l = \left[-\sum_{i=l+1}^{n} s_{1,i}\right] E_1, \ldots, E_{k-1}^l = \left[-\sum_{i=l+1}^{n} s_{k-1,i}\right] E_{k-1}$$

and the associated ZK proof can hence be simulated as well. At the end of the round robin protocol, the parties output the correct public key $E_1, \ldots, E_{k-1}$.

If $\mathcal{A}$ deviates from the protocol in any way, this is caught be the ZK proofs π_i^2 and $\mathcal{S}$ will be able to abort. Thus the protocol, assuming no abort occurs, will output the same public keys as provided by the ideal functionality.

B.3 Security Proofs of the Full-TSS Scheme

We prove the security of our Full-TSS presented in Figure 16 against an active, static adversary, controlling up to $n-1$ parties. Having a full-threshold access structure we assume that the only honest party is party P_j. Since we are in the dishonest majority setting, we are only able to prove security with aborts, which means that if the parties detect an anomaly, then they decide to abort the protocol.

The proof is based on the simulation paradigm. Specifically, we define an ideal functionality $\mathcal{F}_{\mathsf{Dist-CSI-SharK}}$ which represents an ideal execution of the distributed key generation and signing algorithm, which takes the inputs from the adversary and the honest party and returns the signature. This is opposed to the real world protocol in Figure 16 where the adversary interacts with the honest party. To prove security we show that there exists an efficient simulator $\mathcal{S}$ that impersonates the honest party in the ideal execution with the functionality such that the adversary cannot tell if it is interacting with the real party in the real protocol or with the simulator in the ideal world. In other words, the simulator needs to make the transcripts of its interaction with the adversary in the ideal world indistinguishable from the transcripts coming from the interaction between the adversary and P_j in the real protocol. In order to do so, the simulator needs to extrqct the adversarial inputs so as to query the ideal functionality on them and get the correct result, from which it can fake the messages of the honest party. This is done via the extraction algorithms built in the proof of special soundness of the ZK arguments, that requires rewinding the adversary.

In Figure 19, we define the functionality $\mathcal{F}_{\mathsf{Dist-CSI-SharK}}$. The functionality works with n parties and consists of two sub-functionalities for the key generation and signing. Then we prove the following theorem.

Theorem B.2. *The protocols in figures 12 and 16 securely implements the functionality $\mathcal{F}_{\mathsf{Dist-CSI-SharK}}$ (defined in Figure 19) in the f_{Commit}-hybrid model against an active adversary corrupting up to $n-1$ parties.*

KeyGen Simulation: Let P_j is the honest party. $\mathcal{A}$ and $\mathcal{S}$ engage in an execution of the *full-threshold $\mathbb{C}_k$-VPwAI generation* protocol (given in Figure 12). As each party needs to commit to its secret share of s, the simulator commits to a random share s_j^*, say $E_{P_j}^* = [s_j^*]E_0$, produces a simulated proof and then commits to the

The functionality works with parties $P_1, \ldots, P_n$ as follows:

KeyGen: Upon receiving $(\mathbb{Z}_N, E_0, \mathbb{C}_{k-1}, \mathsf{Init})$ from all parties:

1. The functionality samples a random $a \leftarrow \mathbb{Z}_N$ and then computes the structured public key with $k-1$ curves as $E_1 := [a]E_0, E_2 := [c_2 a]E_0, \cdots, E_{k-1} := [c_{k-1}a]E_0$.
2. The adversary enters shares $a_1, \ldots, a_{\hat{j}}, \ldots, a_n$.
3. The functionality completes the adversarial shares with an honest share a_j. Specifically, it computes $a_j := a - \sum_{i \neq j} a_i$.
4. The functionality sends $E_1, \ldots, E_{k-1}$ to the adversary and waits for input from the adversary. If the input is abort then aborts execution. Otherwise sends the public keys to the honest party along with the honest share of the secret key and stores $(\mathbb{Z}_N, E_0, \mathbb{C}_{k-1}, \mathsf{Init}; a)$ internally.

Sign: On input a message m from all parties, the functionality proceeds as follows:

1. The functionality 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 as in a normal CSI-SharK signature using the information in $(\mathbb{Z}_N, E_0, \mathbb{C}_{k-1}, \mathsf{Init}; a)$.
3. The signature is returned to the adversary, and the functionality again waits for input. If the input is again not abort then the functionality returns σ to the honest players.

Fig. 19. Distributed Signature Functionality: $\mathcal{F}_{\mathsf{Dist-CSI-SharK}}$

curve and proof using the commitment scheme. If later it is asked to open this, it will simply equivocate the commitment so that it can be opened to the correct elliptic curve and proof that it is able to compute using the adversarial shares. From the $\pi^1_{P_i}$, given in the commit phase, $\mathcal{S}$ is able to extract the values s_i entered by $\mathcal{A}$ in the first round of proofs. The extracted values s_i are now passed to the functionality, which completes them to a valid set of shares of the secret and returns the corresponding public key $E_0, E_1 = [s]E_0, \ldots, E_{k-1} = [c_{k-1}s]E_0$. At this point, $\mathcal{S}$ has all the adversarial shares and the public key. The honest share s_j is unknown to $\mathcal{S}$. Even though it does not have the honest share, it can fake the commitment by setting

$$E_{P_j} := \left[-\sum_{i \neq j} s_i\right] E_1$$

which it can do by using the public key it got from the functionality and the adversarial shares that it got from the PoKs. Having E_0 and E_{P_j}, $\mathcal{S}$ can simulate the corresponding proof. It then commits to this proof using $\mathcal{F}^{P_j}_{\mathsf{Commit}}$. The commitments can now be opened. Now $\mathcal{A}$ and $\mathcal{S}$ proceed with the round-robin protocol for computing the public keys as in step 8 of Figure 12. All steps for honest players can be simulated exactly by following the real protocol, except that for the party P_j which holds the unknown share s_j. The input to this party in execution j will be

$$E_1^{j-1} = \left[\sum_{i=1}^{j-1} s_i\right] E_0, \ldots, E_{k-1}^{j-1} = \left[\sum_{i=1}^{j-1} c_{k-1}s_i\right] E_0$$

while the output needs to be

$$E_1^j = \left[-\sum_{i=j+1}^{n} s_i\right] E_1, \ldots, E_{k-1}^j = \left[-\sum_{i=j+1}^{n} c_{k-1} s_i\right] E_{k-1}$$

so as to create the correct output public keys $E_1, \ldots, E_{k-1}$. The curves $E_1^j, \ldots, E_{k-1}^j$ can thus be computed by $\mathcal{S}$ like it did for computing E_{P_j} and the associated ZK proof can hence be simulated as well. If $\mathcal{A}$ deviates from the protocol in any way, this is caught be the ZK proofs and $\mathcal{S}$ will be able to abort. Thus the protocol, assuming no abort occurs, will output the same public keys as provided by the ideal functionality.

Sign Simulation: The signing simulation is roughly the same as the key generation simulation. The adversarial inputs can be derived from the initial commitments in the *full-threshold k-MT-GAIP generation protocol* in Figure 9, for $k = 2$. In our simulation of *full-threshold 2-MT-GAIP generation protocol* the value b_j is unknown and 'fixed' by the implicit equation given by the signature (r, c_d) returned by the functionality which gives us $E' = [b]E_0 = [r]E_{c_d}$.

The final part of the signature which needs to be simulated is the output of r_j. We know what $\mathcal{A}$ *should* output and hence can define $r_j = r - \sum_{i \neq j} r_i$. If $\mathcal{A}$ 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. □

B.4 Proof of Theorem 5.2

Proof. This proof closely follows the proof of Theorem 3 in [BDPV21], which proves the consistency and secrecy of the original CSI-RAShi scheme. The authors of [BDPV21] reduce consistency to the soundness of both the PVP and of the NIZK for $j = 1$, while secrecy follows from the zero-knowledge property of the PVP and the NIZK for $j = 1$, as well as from the decisional GAIP assumption (Definition 2.2). The decisional GAIP is needed in the simulation-based proof to ensure that the adversary cannot distinguish between the real and simulated transcripts. While we are leaving the PVP unchanged, we use the NIZK for the case $j = k - 1$, so we have to adapt the proof accordingly. To this end, we can readily use the results from Theorem 3.3, which implies these properties. By further assuming the $\mathbb{C}_{k-1}$-Decisional GAIP, we can also ensure the indistinguishability of the real and simulated transcripts. □