Bandwidth-efficient threshold EC-DSA

Guilhem Castagnos[1], Dario Catalano[2], Fabien Laguillaumie[3], Federico Savasta[2,4], and Ida Tucker[3]

[1] Université de Bordeaux, INRIA, CNRS, IMB UMR 5251, F-33405 Talence, France.
[2] Università di Catania, Italy.
[3] Univ Lyon, EnsL, UCBL, CNRS, Inria, LIP, F-69342, LYON Cedex 07, France.
[4] Scuola Superiore di Catania, Italy

Abstract. Threshold Signatures allow n parties to share the power of issuing digital signatures so that any coalition of size at least $t+1$ can sign, whereas groups of t or less players cannot. Over the last few years many schemes addressed the question of realizing efficient threshold variants for the specific case of EC-DSA signatures. In this paper we present new solutions to the problem that aim at reducing the overall bandwidth consumption. Our main contribution is a new variant of the Gennaro and Goldfeder protocol from ACM CCS 2018 that avoids all the required range proofs, while retaining provable security against malicious adversaries in the dishonest majority setting. Our experiments show that – for all levels of security – our signing protocol reduces the bandwidth consumption of best previously known secure protocols for factors varying between 4.4 and 9, while key generation is consistently two times less expensive. Furthermore compared to these same protocols, our signature generation is faster for 192-bits of security and beyond.

1 Introduction

A threshold signature scheme allows n, mutually mistrusting, users to share the capability of signing documents under a common public key. The threshold $t < n$ typically indicates that any subset of at least $t + 1$ users can collaborate in order to issue a valid signature. On the other hand, no coalition of t or less users can do so. Moreover, if an attacker corrupts up to t users this does not leak any information on the underlying secret key. This latter property is very useful in practice as it significantly reduces the loss induced by a security break in. The study of threshold signatures (and more generally of threshold cryptography [Des88,DF90,GJKR96b,SG98,Sho00,Boy86,CH89,MR04]) attracted significant interest from the early 1990s to the early 2000s. Over the last few years, threshold signatures and, in particular, threshold EC-DSA signatures raised renewed interest. This mainly comes from the fact that EC-DSA is the signature scheme adopted in Bitcoin and other cryptocurrencies. Indeed, a secure, flexible and efficient protocol for threshold EC-DSA signatures can be very effective against the theft of Bitcoins. Protecting EC-DSA signing keys is equivalent to securing Bitcoin: instead of storing a signing key in one single location one could share it among several servers so that none of them knows it in full and a quorum is

needed to produce new signatures. This also means that an attacker should be able to break in into more than t servers to get anything meaningful.

Notice that, in order for a secure solution to be any useful in the cryptocurrency world, efficiency and flexibility are of fundamental importance. Here flexibility mainly refers to the possibility of arbitrarily setting the threshold. Efficiency, on the other hand, takes into account both the computational costs and the bandwidth consumption induced by the protocol.

Before the advent of cryptocurrencies, known solutions to the problem fell short either in terms of flexibility or in terms of efficiency (or both). The state of the art was the work of Gennaro *et al.* [GJKR96a] where to implement a threshold of t servers one needed to share the key among a total of at least $n = 2t + 1$ servers, thus making n-out-of-n sharings impossible (i.e. sharings where all parties are required to participate to the signing process). This was later addressed by Mackenzie and Reiter [MR01] for the specific two party setting (i.e. where $t = 1$ and $n = 2$) but the proposed protocol heavily relies on inefficient zero knowledge proofs, thus making the resulting protocol of little practical interest.

Over the last few years, improved solutions have been proposed both for the two party [Lin17,DKLs18,CCL$^+$19] and for the more general t-out-of-n case [GGN16,GG18,LN18,DKLs19]). Focusing on the latter case, all these solutions still have drawbacks either in terms of bandwidth costs (e.g. [DKLs19] and [LN18] for their OT implementation), somewhat heavy setup ([GGN16]) or underlying assumptions ([GG18]).

Our contribution. In this paper we present new techniques to realize efficient threshold variants of the EC-DSA signature scheme. Our resulting protocols are particularly efficient in terms of bandwidth consumption and, as several recent works (e.g. [GG18]) allow to consider any threshold t such that $n \geq t + 1$.

Our main contribution is a new variant of the Gennaro and Goldfeder protocol [GG18] that manages to avoid all the required range proofs, while retaining comparable overall (computational) efficiency.

To better explain our contribution let us briefly describe how (basic) EC-DSA works. The public key is an elliptic curve point Q and the signing key is x, where $Q \leftarrow xP$, and P is a generator of the group of points of the elliptic curve of prime order q. To sign a message m one first hashes it using some hash function H and then proceeds as follows. Choose a random $k \in \mathbf{Z}/q\mathbf{Z}$ and compute $R = k^{-1}P$. Letting $r \leftarrow r_x \bmod q$ – where $R = (r_x, r_y)$ – set $s \leftarrow k(H(m) + rx) \bmod q$. The signature is the pair (r, s).

The difficulty when trying to devise a threshold variant of this scheme comes from the fact that one has to compute both $R = k^{-1}P$ and a multiplication of the two secret values k, x. In [GG18] Gennaro and Goldfeder address this as follows. Starting from two secrets $a = a_1 + \cdots + a_n$, $b = b_1 + \cdots + b_n$ additively shared among the parties (i.e. P_i holds a_i and b_i), players compute $ab = \sum_{i,j} a_i b_j$ by computing additive shares of each $a_i b_j$. This can be achieved via a simple two party protocol, originally proposed by Gilboa [Gil99] in the setting of two party RSA key generation, which parties execute in a pairwise way. Slightly more in detail, this latter protocol relies on linearly homomorphic encryption and Gennaro

and Goldfeder implement it using Paillier's cryptosystem as underlying building block. This choice, however, becomes problematic when dealing with malicious adversaries, as Paillier plaintexts live in $(\mathbf{Z}/N\mathbf{Z})$ (for N large composite) whereas EC-DSA signatures live in $\mathbf{Z}/q\mathbf{Z}$ (q prime). To avoid inconsistencies, one then needs to choose N significantly larger than q, so that no wrap arounds occur during the execution of the whole protocol. To prevent malicious behavior, this also induces the need of expensive range proofs, i.e. when sending $\mathsf{Enc}(x_i)$ a player also needs to prove that x_i is small enough.

To fix this, one might be tempted to resort to the hash proof systems based technique recently proposed by Castagnos *et al.* [CCL+19]. This methodology allows an efficient instantiation from class groups of imaginary quadratic fields that, in turn, builds upon the Castagnos and Laguillaumie [CL15] homomorphic encryption scheme. One key feature of this scheme and its variants (CL from now on) is that they allow instantiations where the message space is $\mathbf{Z}/q\mathbf{Z}$ and this q can be the same large prime used in EC-DSA signatures. Unfortunately, however, this feature comes at the cost of loosing surjectivity. More precisely, and differently than Paillier, CL is not surjective in the ciphertext space and the set of valid CL ciphertexts is not even efficiently recognizable. Even worse, known techniques to prove the validity of a CL ciphertext are rather inefficient as they all use binary challenges. This means that to get soundness error 2^{-t} the proof needs to be repeated t times.

Back to our threshold EC-DSA setting, naively switching from Paillier to CL, only means trading inefficient range proofs with inefficient proofs of validity for ciphertexts!

In this paper, we develop new techniques that address exactly this issue. As a first contribution we develop new efficient protocols to prove CL ciphertexts are well formed. This result is quite general and can have useful applications even beyond the specific threshold setting considered in this paper (and indeed can be used to improve the efficiency of the recent two party protocol from [CCL+19]).

Next, we revisit the Gennaro and Goldfeder protocol and propose a new CL-based EC-DSA variant where the aforementioned multiplication step can be done efficiently and without resorting to range proofs.

Our constructions rely on two recently introduced assumptions on class groups. Informally, given a group $\widehat{G}$ the first one states that it is hard to find low order elements in $\widehat{G}$ (low order assumption) while the latter assumes that it is hard to find roots of random elements in $\widehat{G}$ (strong root assumption). Both these assumptions are believed to hold in class groups of imaginary quadratic fields ([BH01,DF02,BBHM02,Lip12]) and were recently used in, e.g. [BBF18,Pie19,Wes19].

From a technical perspective, resorting to these assumptions allows us to dramatically improve the efficiency of the (zero knowledge) arguments of knowledge needed by our protocols. Informally this can be explained as follows. In the class group setting, the order of the group $\widehat{G}$ is unknown (to all parties, even to those who set up the parameters). This is typically a bad thing when doing arguments of knowledge as, unless one restricts to binary challenges, it is not immediate how to argue the extractability of the witness.

In our proofs, we manage to prove that, no matter how big the challenge space is, either one can extract the witness or one can find a root for some given (random) element of the group, thus violating the strong root assumption. Our argument is actually more convoluted than that as, for technical reasons that won't be discussed here, we still need to make sure that no undetected low order elements are maliciously injected in the protocols (e.g. to extract unauthorized information). This is where the low order assumption comes into play and allows us to avoid hard to handle corner cases in our proofs. Challenges also arise from the fact that in order to reduce to the hardness of finding roots, our reduction should output e^{th} roots where e is not a power of two, since, as observed in concluding remarks of [CCL+19], computing square roots or finding elements of order 2 can be done efficiently in class groups knowing the factorization of the discriminant (which is public in our case).

We also provide in Section 5 a zero knowledge proof of knowledge (without computational assumptions) for groups of unknown order in order to improve our setup. That proof can also be of independent interest and actually improves the key generation of [CCL+19] for two party EC-DSA.

Efficiency comparisons. We compare the speed and communication costs of our protocol to those of the scheme by Gennaro and Goldfeder [GG18] and that of Lindell *et al.* [LN18] for the standard NIST curves P-256, P-384 and P-521, corresponding to levels of security 128, 192 and 256. For the encryption scheme, we start with a 112 bit security, as in their implementations, but also study the case where its level of security matches that of the elliptic curves. Our comparisons show that for all security levels our signing protocol reduces the bandwidth consumption of best previously known secure protocols for factors varying between 4.4 and 9, while key generation is consistently two times less expensive. Moreover, we even outperform (for all security levels) the stripped down implementation of [GG18] where a number of range proofs are omitted. We believe this to be an important aspect of our schemes. Indeed, as Gennaro and Goldfeder themselves point out in [GG18], omitting these proofs leaks information on the shared signing key. While they conjecture that this information is limited enough for the protocol to remain secure, no formal analysis is provided.

In terms of timings, though for standard levels of security (112 and 128) our signing protocol is up to four times slower than that of [LN18], for higher levels of security the trend is inverted, such that for 256-bit security we are twice as fast as all other secure schemes considered[5].

2 Preliminaries

Notations. For a distribution $\mathcal{D}$, we write $d \hookleftarrow \mathcal{D}$ to refer to d being sampled from $\mathcal{D}$ and $b \xleftarrow{\$} B$ if b is sampled uniformly in the set B. In an interactive protocol IP, between parties $P_1, \ldots, P_n$ for some integer $n > 1$, we denote by

[5] But still twice as slow as the stripped down [GG18] protocol

$\mathsf{IP}\langle x_1;\ldots;x_n\rangle \to \langle y_1;\ldots;y_n\rangle$ the joint execution of parties $\{P_i\}_{i\in[n]}$ in the protocol, with respective inputs x_i, and where P_i's private output at the end of the execution is y_i. If all parties receive the same output y we write $\mathsf{IP}\langle x_1;\ldots;x_n\rangle \to \langle y\rangle$. A (P)PT algo stands for an algorithm running in (probabilistic) polynomial time w.r.t. the length of its inputs.

2.1 Tools

Zero-knowledge proofs. A zero-knowledge proof of knowledge (ZKPoK) system for a binary relation R is an interactive protocol (P, V) between two probabilistic algorithms: a prover P and a PT verifier V. Informally P, detaining a witness w for a given statement x s.t. $(x, w) \in \mathsf{R}$, must convince V that it is true without revealing anything other to V. In a zero-knowledge argument of knowledge (ZKAoK), the proof provided by P is computationally sound (P is also a PT algorithm). Formal definitions for computationally convincing proofs of knowledge are provided in the next paragraph. We use the notation introduced by Camenisch-Stadler [CS97], which conveniently expresses the goals of a ZKP (resp. ZKA) scheme:

$$\mathsf{ZKPoK}_x\{(w) : (x, w) \in \mathsf{R}\} \quad \text{and} \quad \mathsf{ZKAoK}_x\{(w) : (x, w) \in \mathsf{R}\}.$$

Computationally convincing proofs of knowledge. We here provide some terminology and definitions relating to computationally convincing proofs of knowledge (or arguments of knowledge) as defined in [DF02]. Consider a *relation generator* algorithm $\mathcal{R}$, that takes as input 1^λ and outputs the description of a binary relation R. A prover is a machine P who gets R as an input, outputs a statement x and finally conducts the interactive proof with a verifier V using R, x as common input. From P, define a machine P_{view} which starts in the state P is in after having seen view view and having produced x. P_{view} then conducts the protocol with V following P's algorithm. The view view contains all inputs, messages exchanged and random coins so in particular x is determined by view. We note $\epsilon_{\mathsf{view},P}$ P's probability to make V accept, conditioned on view. The knowledge error function $\kappa(\lambda)$ is the probability that P can make V accept without knowing a witness w s.t. $(x, w) \in \mathsf{R}$ (for a security parameter λ). An extractor is a machine M that gets R and a statement x as an input, has black-box access to P_{view} for some view consistent with x and computes a witness w s.t. $(x, w) \in \mathsf{R}$.

Definition 1. *For some given cheating P^*, extractor M and polynomial $p()$, M fails on view view if $\epsilon_{\mathsf{view},P} > \kappa(\lambda)$, and the expected running time of M using P^*_{view} as oracle, is greater than $\frac{p(\lambda)}{\epsilon_{\mathsf{view},P^*} - \kappa(\lambda)}$.*

Definition 2. *Let $\mathcal{R}$ be a probabilistic polynomial time relation generator, and consider a protocol (P, V), a knowledge extractor M, polynomial $p()$ and knowledge error function $\kappa(\lambda)$ be given. Consider the following experiment with input λ: $\mathsf{R} := \mathcal{R}(1^\lambda)$, $x := P^*(\mathsf{R})$ which defines view view. The advantage of P^*, denoted $\mathsf{Adv}_{\kappa,M,p}(P^*, \lambda)$, is the probability (taken over the random coins of $\mathcal{R}$, P^*) that M fails on the view generated by this experiment.*

Definition 3. *Let $\mathcal{R}$ be a PPT relation generator. (P, V) is a computationally convincing proof of knowledge for $\mathcal{R}$, with knowledge error $\kappa()$, failure probability $\nu()$ and time bound $t()$, if the following hold:*
Completeness: *The honest prover P receives $\mathsf{R} \leftarrow \mathcal{R}(1^\lambda)$, produces $(x, w) \in \mathsf{R}$, sends x to V and finally conducts the protocol with V, who accepts with overwhelming probability in λ.*
Soundness: *There exists a polynomial $p()$ and an extractor M, s.t. for all provers P^* running in time at most $t(\lambda)$, $\mathsf{Adv}_{\kappa,M,p}(P^*, \lambda) \leq \nu(\lambda)$.*

Threshold secret sharing. A (t, n) threshold secret sharing scheme allows to divide a secret s into shares $s_1, \ldots, s_n$, amongst a group of n participants, in such a way that knowledge of any $t + 1$ or more shares allows to compute s; whereas knowledge of any t or less shares reveals no information about s.

Feldman verifiable secret sharing. A verifiable secret sharing (VSS) protocol allows to share a secret between n parties $P_1, \ldots, P_n$ in a verifiable way. Specifically, it can be used by a party to share a secret with the other ones. Feldmann's VSS [Fel87] relies on Shamir's secret sharing scheme [Sha79], but the former gives additional information allowing to check the sharing is done correctly.

Let $\mathbb{G}$ be a group of order q, g a generator of $\mathbb{G}$, and suppose that one of the players, that we call P, wants to share a secret $\sigma \in \mathbf{Z}/q\mathbf{Z}$ with the other ones. To share the secret, it does the following steps:

1. P generates a random polynomial $p \in \mathbf{Z}/q\mathbf{Z}[x]$ of degree t and with σ as free term. The polynomial is then
$$p(x) = a_t x^t + a_{t-1} x^{t-1} + \ldots + a_2 x^2 + a_1 x + \sigma \mod q,$$
where $\sigma = p(0) \mod q$. The shares of σ are $\sigma_i = p(i) \mod q$.
2. P sends σ_i to P_i, for all i.
3. P publishes auxiliary information that other players can use to check the shares are consistent and define a unique secret: $\{v_i = g^{a_i} \in \mathbb{G}\}_{i \in [t]}$ and $v_0 = g^\sigma \in \mathbb{G}$.

Each party can check its own share is consistent by verifying if the following condition holds:
$$g^{\sigma_i} = \prod_{j=0}^{t} v_j^{i^j} \in \mathbb{G}$$

If one of the checks fails, then the protocol terminates. Furthermore, the only information that the Feldman's VSS leaks about the secret σ is $v_0 = g^\sigma$.

Commitments. An equivocable commitment scheme allows a sender S to commit to a message m s.t. S's message is perfectly hidden; in the opening phase – where S reveals m and an opening value $\mathsf{d}(m)$ to R – S is computationally bound to the committed message. Consequently the scheme allows for a trapdoor which allows to open a commitment to arbitrary messages (this is called equivocating the commitment). The trapdoor should be hard to compute efficiently.

Formally a (non-interactive) equivocable commitment scheme consists of four PPT algorithms: (1) $\mathsf{Setup}(1^\lambda) \to (\mathsf{pp}, \mathsf{tk})$ which outputs public parameters pp and associated secret trapdoor key tk; (2) $\mathsf{Com}(m, r) \to [\mathsf{c}(m), \mathsf{d}(m)]$ which on input a message m and random coins r, outputs the commitment $\mathsf{c}(m)$ and an opening value $\mathsf{d}(m)$ (if S refuses to open a commitment $\mathsf{d}(m) = \perp$); (3) $\mathsf{Open}(\mathsf{c}, \mathsf{d}) \to m$ or $\perp$ which on input a commitment c and an opening value d, outputs either a message m or an error symbol $\perp$; (4) $\mathsf{Equiv}(\mathsf{tk}, m, r, m') \to \widehat{\mathsf{d}}$ which – if tk is a trapdoor key for pp – allows to open commitments $\mathsf{c}(m)$ to arbitrary values m'. Precisely, for any messages m and m', any $\mathsf{Setup}(1^\lambda) \to (\mathsf{pp}, \mathsf{tk})$, let $\mathsf{Com}(m, r) \to [\mathsf{c}(m), \mathsf{d}(m)]$ and $\mathsf{Equiv}(\mathsf{tk}, m, r, m') \to \widehat{\mathsf{d}}$ then $\mathsf{Open}(\mathsf{c}(m), \widehat{\mathsf{d}}) \to m'$; and s.t. opening fake and real commitments is indistinguishable. We will use equivocable commitments with the following properties:
Correctness: for all message m and randomness r, if $[\mathsf{c}(m), \mathsf{d}(m)] \leftarrow \mathsf{Com}(m, r)$, one has $m \leftarrow \mathsf{Open}(\mathsf{c}(m), \mathsf{d}(m))$.
Perfect hiding: for every message pair m, m' the distributions of the resulting commitments are statistically close.
Computational binding: for any PPT algorithm $\mathcal{A}$, the probability that $\mathcal{A}$ outputs $(\mathsf{c}, \mathsf{d}_0, \mathsf{d}_1)$ s.t. $\mathsf{Open}(\mathsf{c}, \mathsf{d}_0) \to m_0$; $\mathsf{Open}(\mathsf{c}, \mathsf{d}_1) \to m_1$; $m_0 \neq \perp$; $m_1 \neq \perp$ and $m_0 \neq m_1$ is negligible in the security parameter.
Concurrent non-malleability: a commitment scheme is non-malleable [DDN00] if no PPT adversary $\mathcal{A}$ can "maul" a commitment to a value m into a commitment to a related value $\overline{m}$. The notion of a concurrent non-malleable commitment [DDN00,PR05] further requires non-malleability to hold even if $\mathcal{A}$ receives many commitments and can itself produce many commitments.

2.2 The elliptic curve digital signature algorithm

Elliptic curve digital signature algorithm. EC-DSA is the elliptic curve analogue of the Digital Signature Algoritm (DSA). It was put forth by Vanstone [Van92] and accepted as ISO, ANSI, IEEE and FIPS standards. It works in a group $(\mathbb{G}, +)$ of prime order q (of say μ bits) of points of an elliptic curve over a finite field, generated by P and consists of the following algorithms.

$\mathsf{KeyGen}(\mathbb{G}, q, P) \to (x, Q)$ where $x \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ is the secret signing key and $Q := xP$ is the public verification key.

$\mathsf{Sign}(x, m) \to (r, s)$ where r and s are computed as follows:

1. Compute m': the μ leftmost bits of $\mathsf{SHA256}(m)$ where m is to be signed.
2. Sample $k \xleftarrow{\$} (\mathbf{Z}/q\mathbf{Z})^*$ and compute $R := k^{-1}P$; denote $R = (r_x, r_y)$ and let $r := r_x \mod q$. If $r = 0$ choose another k.
3. Compute $s := k \cdot (m' + r \cdot x) \mod q$.

$\mathsf{Verif}(Q, m, (r, s)) \to \{0, 1\}$ indicating whether or not the signature is accepted.

The standard security notion required of digital signature schemes is that of existential unforgeability under chosen message attacks (eu-cma) [GMR88].

Definition 4 (Existential unforgeability [GMR88]). *Consider a digital signature scheme* $\mathsf{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verif})$*, and a PPT algorithm* $\mathcal{A}$*, which is given as input a verification key* vk *output by* $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{sk}, \mathsf{vk})$ *and oracle access to the signing algorithm* $\mathsf{Sign}(\mathsf{sk}, .)$ *to whom it can (adaptively) request signatures on messages of its choice. Let* $\mathcal{M}$ *be the set of queried messages.* S *is existentially unforgeable if for any such* $\mathcal{A}$*, the probability* $\mathsf{Adv}^{\mathsf{eu\text{-}cma}}_{\mathsf{S},\mathcal{A}}$ *that* $\mathcal{A}$ *produces a valid signature on a message* $m \notin \mathcal{M}$ *is a negligible function of* λ*.*

(t, n)-threshold EC-DSA. For a threshold t and a number of parties $n > t$, threshold EC-DSA consists of the following interactive protocols:

$\mathsf{IKeyGen}\langle(\mathbb{G}, q, P); \ldots; (\mathbb{G}, q, P)\rangle \to \langle(x_1, Q); \ldots; (x_n, Q)\rangle$ s.t. $\mathsf{KeyGen}(\mathbb{G}, q, P) \to (x, Q)$ where the values $x_1, \ldots, x_n$ constitute a (t, n) threshold secret sharing of the signing key x.

$\mathsf{ISign}\langle(x_1, m); \ldots; (x_n, m)\rangle \to \langle(r, s)\rangle$ **or** $\langle\perp\rangle$ where $\perp$ is the error output, signifying the parties may abort the protocol, and $\mathsf{Sign}(x, m) \to (r, s)$.

The verification algorithm is non interactive and identical to that of EC-DSA.

Following [GJKR96b], we present a game-based definition of security analogous to $\mathsf{eu\text{-}cma}$: threshold unforgeability under chosen message attacks ($\mathsf{tu\text{-}cma}$).

Definition 5 (Threshold signature unforgeability [GJKR96b]). *Consider a* (t, n)*-threshold signature scheme* $\mathsf{IS} = (\mathsf{IKeyGen}, \mathsf{ISign}, \mathsf{Verif})$*, and a PPT algorithm* $\mathcal{A}$*, having corrupted at most* t *players, and which is given the view of the protocols* $\mathsf{IKeyGen}$ *and* ISign *on input messages of its choice (chosen adaptively) as well as signatures on those messages. Let* $\mathcal{M}$ *be the set of aforementioned messages.* IS *is unforgeable if for any such* $\mathcal{A}$*, the probability* $\mathsf{Adv}^{\mathsf{tu\text{-}cma}}_{\mathsf{IS},\mathcal{A}}$ *that* $\mathcal{A}$ *can produce a signature on a message* $m \notin \mathcal{M}$ *is a negligible function of* λ*.*

2.3 Building blocks from Class Groups

An instantiation of the CL *framework.* Castagnos and Laguillaumie introduced the framework of a group with an easy discrete logarithm (Dlog) subgroup in [CL15], which was later enhanced in [CLT18,CCL+19] and gave concrete instantiation from class groups of quadratic fields. Some background on class groups of quadratic fields in cryptography can be found in [BH01] and in [CL15, Appx. B].

We briefly sketch the instantiation given in [CCL+19, Sec. 4.1] and the resulting group generator Gen that we will use in this paper. The interested reader can refer to [CL15,CCL+19] for concrete details.

Given a prime q consider another random prime $\tilde{q}$, the fundamental discriminant $\Delta_K = -q\tilde{q}$ and the associated class group $C(\Delta_K)$. By choosing $\tilde{q}$ s.t. $q\tilde{q} \equiv -1 \pmod 4$ and $(q/\tilde{q}) = -1$, we have that the $2-$Sylow subgroup of $C(\Delta_K)$ has order 2. The size of $\tilde{q}$ is chosen s.t. computing the class number $h(\Delta_K)$ takes time 2^λ. We then consider the suborder of discriminant $\Delta_q = -q^2\Delta_K$. Then, we denote $(\widehat{G}, \cdot)$ the finite abelian subgroup of squares of $C(\Delta_q)$, which corresponds to the odd part. It is possible to check efficiently if an element is in $\widehat{G}$

(cf. [Lag80]). One can exhibit a subgroup F generated by $f \in \widehat{G}$ where f is represented by an ideal of norm q^2. This subgroup has order q and there exists a deterministic PT algorithm for the discrete logarithm (Dlog) problem in F (cf. [CL15, Proposition C – 1]). Then we build deterministically a $q-$th power of $\widehat{G}$ by lifting the class of an ideal of discriminant Δ_K above the smallest splitting prime. In the following, we will denote $\hat{g}_q$ this deterministic generator. We will then consider an element g_q constructed as a random power of $\hat{g}_q$. This slightly changes the construction of [CCL$^+$19], in order to make a reduction to a strong root problem for the soundness of the argument of knowledge of Subsection 3.1. One can compute an upper bound $\tilde{s}$ for the order of $\hat{g}_q$, using an upper bound of $h(\Delta_K)$. For this, one can use the fact that $h(\Delta_K) < \frac{1}{\pi}\log|\Delta_K|\sqrt{|\Delta_K|}$, or obtain a slightly better bound from the analytic class number formula.

For our application the prime q will have at least 256 bits, in that case q is prime to $h(\Delta_K)$ except with negligible probability. Therefore q will be prime to the order of $\hat{g}_q$ which is a divisor of $h(\Delta_K)$.

Notation. We denote Gen the algorithm that on input a security parameter λ and a prime q, outputs $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F)$ defined as above. We also denote Solve the deterministic PT algorithm that solves the Dlog problem in F. This pair of algorithms is an instance of the framework of a group with an easy Dlog subgroup (cf. [CCL$^+$19, Definition 4]). For a random power g_q of $\hat{g}_q$ we will denote G^q the subgroup generated by g_q, $g = g_q f$ and G the subgroup generated by g.

Hard subgroup membership assumption. We recall the definition of the HSM problem for an output $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F)$ of Gen. For a random power g_q of $\hat{g}_q$ the HSM assumption states it is hard to distinguish the elements of G^q in G. As a result this HSM assumption is closely related to Paillier's DCR assumption, they are essentially the same assumption in different groups, hence there is no direct reduction between them. HSM was first used by [CLT18] within class groups, though cryptography based on class groups is now well established, and is seeing renewed interest (e.g. [CIL17,CLT18,BBBF18,Wes19,CCL$^+$19]).

Definition 6 (HSM assumption). *For $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F)$ an output of Gen, g_q a random power of $\hat{g}_q$ and $g := g_q f$, we denote $\mathcal{D}$ (resp. $\mathcal{D}_q$) a distribution over the integers s.t. the distribution $\{g^x, x \hookleftarrow \mathcal{D}\}$ (resp. $\{\hat{g}_q^x, x \hookleftarrow \mathcal{D}_q\}$) is at distance less than $2^{-\lambda}$ from the uniform distribution in $\langle g\rangle$ (resp. in $\langle \hat{g}_q\rangle$). Let $\mathcal{A}$ be an adversary for the HSM problem, its advantage is defined as:*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{HSM}}(\lambda) := \Big| 2 \cdot \Pr\big[b = b^\star : (\tilde{s}, f, \hat{g}_q, \widehat{G}, F) \leftarrow \mathsf{Gen}(1^\lambda, q), t \hookleftarrow \mathcal{D}_q, g_q = \hat{g}_q^t, x \hookleftarrow \mathcal{D}, x' \hookleftarrow \mathcal{D}_q, b \xleftarrow{\$} \{0,1\}, Z_0 \leftarrow g^x, Z_1 \leftarrow g_q^{x'}, b^\star \leftarrow \mathcal{A}(q, \tilde{s}, f, \hat{g}_q, g_q, \widehat{G}, F, Z_b, \mathsf{Solve}(.))\big] - 1 \Big|$$

The HSM problem is said to be hard in G if for all probabilistic polynomial time algorithm $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{HSM}}(\lambda)$ is negligible.

Remark that compared to previous works, we modify slightly the assumption by considering a random element g_q instead of using the deterministic element $\hat{g}_q$.

A linearly homomorphic encryption scheme. We recall the linearly homomorphic encryption scheme of [CLT18] whose ind-cpa-security relies on the HSM assumption. The scheme somewhat generalises Camenisch and Shoup's approach in [CS03]. This scheme is the basis of the threshold EC-DSA protocol of Sec. 3.

We use the output of $\mathsf{Gen}(1^\lambda, q)$ and as in Def. 6, we set $g_q = \hat{g}_q^t$ for $t \hookleftarrow \mathcal{D}_q$. The public parameters of the scheme are $\mathsf{pp} := (\tilde{s}, f, \hat{g}_q, g_q, \widehat{G}, F, q)$. To instantiate $\mathcal{D}_q$, we set $\tilde{A} \geq \tilde{s} \cdot 2^{40}$ s.t. $\{g_q^r, r \hookleftarrow [\tilde{A}]\}$ is at distance less than 2^{-40} from the uniform distribution in G^q. The plaintext space is $\mathbf{Z}/q\mathbf{Z}$. The scheme is depicted in Fig. 1.

Theorem 1 ([CLT18]). *The* CL *scheme described in Fig. 1 is semantically secure under chosen plaintext attacks (*ind-cpa*) under the* HSM *assumption.*

Algo. $\mathsf{KeyGen}(\mathsf{pp})$

1. Pick $\mathsf{sk} \hookleftarrow [\tilde{A}]$ and $\mathsf{pk} := g_q^{\mathsf{sk}}$
2. Return $(\mathsf{pk}, \mathsf{sk})$

Algo. $\mathsf{Enc}(\mathsf{pk}, m)$

1. Pick $r \hookleftarrow [\tilde{A}]$
2. Return $(g_q^r, f^m \mathsf{pk}^r)$

Algo. $\mathsf{Dec}(\mathsf{sk}, (c_1, c_2))$

1. Compute $M = c_2 / c_1^{\mathsf{sk}}$
2. Return $\mathsf{Solve}(M)$

Fig. 1: Description of the CL encryption scheme

2.4 Algorithmic assumptions

We here provide further definitions for the algorithmic assumptions on which the security of our protocol relies. As in [CCL+19], we need the HSM assumption guaranteeing the ind-cpa-security of the linearly homomorphic encryption scheme. We also use two additional assumptions: one which states that it is hard to find low order elements in the group $\widehat{G}$, and one which states that it is hard to find roots in $\widehat{G}$ of random elements of the subgroup $\langle \hat{g}_q \rangle$. These assumptions allow us to significantly improve the efficiency of the ZKAoK needed in our protocol. Indeed, as the order of the group we work in is unknown, we cannot (unless challenges are binary as done in [CCL+19]) immediately extract the witness from two answers corresponding to two different challenges of a given statement. However we show in the ZKAoK of Sec. 3.1 that whatever the challenge space, if one cannot extract the witness, then one can break at least one of these two assumptions. Consequently these assumptions allow us to significantly increase the challenge space of our proofs, and reduce the number of rounds in the protocol to achieve a satisfying soundness, which yields an improvement both in terms of bandwidth and of computational complexity.

Using such assumptions in the context of generalized Schnorr Proofs in groups of unknown order is not novel (*cf.* e.g. [DF02,CKY09]). We adapt these techniques for our specific subgroups of a class group of an imaginary quadratic field, and state them with respect to Gen.

Definition 7 (Low order assumption). *Consider a security parameter $\lambda \in \mathbf{N}$, and $\gamma \in \mathbf{N}$. The* γ-low order problem *(LOP_γ) is $(t(\lambda), \epsilon_{\mathsf{LO}}(\lambda))$-secure for* Gen *if, given the output of* Gen*, no algorithm $\mathcal{A}$ running in time $\leq t(\lambda)$ can output a γ-low order element in $\widehat{G}$ with probability greater than $\epsilon_{\mathsf{LO}}(\lambda)$. More precisely,*

$$\epsilon_{\mathsf{LO}}(\lambda) := \Pr[\mu^d = 1, 1 \neq \mu \in \widehat{G}, 1 < d < \gamma \ : \\ (\tilde{s}, f, \hat{g}_q, \widehat{G}, F) \xleftarrow{\$} \mathsf{Gen}(1^\lambda, q); (\mu, d) \xleftarrow{\$} \mathcal{A}(\tilde{s}, f, \hat{g}_q, \widehat{G}, F)].$$

The γ-low order assumption holds if $t = poly(\lambda)$, and ϵ_{LO} is negligible in λ.

We now define a strong root assumption for class groups. This can be seen as a generalisation of the strong RSA assumption. We specialise this assumption for class groups where computing square roots is easy knowing the factorisation of the discriminant, and tailor it to our needs by considering challenges in a subgroup.

Definition 8 (Strong root assumption for Class Groups). *Consider a security parameter $\lambda \in \mathbf{N}$, and let $\mathcal{A}$ be a probabilistic algorithm. We run* Gen *on input $(1^\lambda, q)$ to get $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F)$ and we give this output and a random $Y \in \langle \hat{g}_q \rangle$ as an input to $\mathcal{A}$. We say that $\mathcal{A}$ solves the strong root problem for class groups (*SRP*) if $\mathcal{A}$ outputs a positive integer $e \neq 2^k$ for all k and $X \in \widehat{G}$, s.t. $Y = X^e$. In particular, the* SRP *is $(t(\lambda), \epsilon_{\mathsf{SR}}(\lambda))$-secure for* Gen *if any adversary $\mathcal{A}$, running in time $\leq t(\lambda)$, solves the* SRP *with probability at most $\epsilon_{\mathsf{SR}}(\lambda)$.*

On the hardness of these assumptions in class groups. For our applications, we will use the strong root assumption and the low order assumption in the context of class groups. These assumptions are not completely novel in this setting: Damgård and Fujisaki ([DF02]) explicitly consider variants of these assumptions in this context. Then, Lipmaa used a strong root assumption in class groups to build accumulators without trusted setup in [Lip12]. Recently, an interactive variant of the strong root assumption was used, still in the context of class groups, by Wesolowski to build verifiable delay functions without trusted setup. Furthermore, the low order assumption is also used to implement Pietrzak's verifiable delay functions with class groups (see [BBF18,Pie19]). In the following, we advocate the hardness of these assumptions in the context of class groups.

The root problem and its hardness was considered in [BH01,BBHM02] in the context of class groups to design signature schemes. It is similar to the RSA problem: the adversary is not allowed to choose the exponent e. These works compare the hardness of this problem with the problem of computing the group order and conclude that there is no better known method to compute a solution to the root problem than to compute the order of the group.

The strong root assumption is a generalisation of the strong RSA assumption. Again, the best known algorithm to solve this problem is to compute the order of the group to be able to invert exponents. For strong RSA this means factoring the modulus. For the strong root problem in class groups, this means computing the

class number, and best known algorithms for this problem have worst complexity than those to factor integers.

Note that we have specialized this assumption for exponents e which are not powers of 2: as mentioned in [CCL$^+$19], one can compute square roots in polynomial time in class groups of quadratic fields, knowing the factorisation of the discriminant (which is public in our setting), cf. [Lag80].

Concerning the low order assumption, we need the $\gamma-$low order problem to be hard in $\widehat{G}$, where γ can be up to 2^{128}. Note that in our instantiation, the discriminant is chosen such that the $2-$Sylow subgroup is isomorphic to $\mathbf{Z}/2\mathbf{Z}$. It is well known that the element of order 2 can be computed from the (known) factorisation of Δ_q. However, we work with the odd part, which is the group of squares in this context, so we do not take this element into account.

Let us see that the proportion of such elements of low order is very low in the odd part. From the Cohen Lenstra heuristics [CL84] the odd part of a class group $C(\Delta)$ of an imaginary quadratic field is cyclic with probability 97.75%. In [HS06], extending the Cohen Lenstra heuristics, it is conjectured that the probability an integer d divides the order $h(\Delta)$ of $C(\Delta)$ is less than:

$$\frac{1}{d} + \frac{1}{d \log d}.$$

As a consequence, if the odd part of $C(\Delta)$ is cyclic then the expected number of elements of order less than γ is less than

$$\sum_{d \leqslant \gamma} \left(\frac{1}{d} + \frac{1}{d \log d} \right) \varphi(d),$$

which can be bounded above by 2γ. For 128 bits of security, our class number will have around 913 bits, so the proportion of elements of order less than 2^{128} is less than 2^{-784}.

Moreover, if the odd part of the class group is non cyclic, it is very likely that it is of the form $\mathbf{Z}/n_1\mathbf{Z} \oplus \mathbf{Z}/n_2\mathbf{Z}$ where $n_2 | n_1$ and n_2 is very small. Still from the Cohen Lenstra heuristics, the probability that the $p-$rank (the number of cyclic factors in the $p-$Sylow subgroup) of the odd part is equal to r is equal to

$$\frac{\eta_\infty(p)}{p^{r^2}\eta_r(p)^2} \quad \text{where} \quad \eta_r(p) = \prod_{k=1}^{r}(1 - p^{-k}).$$

If we have two cyclic factors, and $p | n_2$, then the $p-$rank is 2. If $p > 2^{20}$ the probability of having a $p-$rank equal to 2 is less than 2^{-80}. Similarly, we cannot have many small cyclic components: the $3-$rank is 6 with probability less than 2^{-83}. Actually, we know only 3 class groups of such 3 ranks [Que87].

There have been intense efforts on the construction of families of discriminants such that there exist elements of a given small order p or with a given $p-$rank. However, these families are very sparse and will be reached by our generation algorithm of the discriminant only with negligible probability. The basic

idea of these constructions is to build a discriminant Δ in order to obtain solutions of a Diophantine equation that gives m and the representation of a non principal ideal I of norm m such that I^p is principal, and I has order p in $C(\Delta)$ (see eg [Bue76] or [Bel04] for more references).

Solving such a norm equation for a fixed discriminant has been mentioned as a starting point for an attack in [BBF18] combined with the Coppersmith's method, but no concrete advances on the problem have been proposed.

3 Threshold EC-DSA protocol

We here provide a construction for (t, n)-threshold EC-DSA signing from the CL framework. Security – which does not degrade with the number of signatures queried by the adversary in the $\mathsf{tu\text{-}cma}$ game (*cf.* Def. 5) – relies on the assumptions and tools introduced in Sec. 2. Throughout the article we consider the group of points of an elliptic curve $\mathbb{G}$ of order q, generated by P.

As in many previous works on multiparty EC-DSA (*e.g.* [MR01,Lin17,GG18]), we use a linearly homomorphic encryption scheme. This enables parties to perform operations collaboratively while keeping their inputs secret. Explicitly a party P_i sends a ciphertext encrypting its secret share (under its own public key) to party P_j, P_j then performs homomorphic operations on this ciphertext (using its own secret share), and sends the resulting ciphertext back to P_i – intuitively P_i should learn nothing more about the operations performed by P_j than that revealed by decrypting the ciphertext it receives. To ensure this, P_i must prove to P_j that the ciphertext it first sent is 'well formed'. To this end in Sec. 3.1, we provide an efficient zero-knowledge argument of knowledge of the plaintext and of the randomness used to compute a CL ciphertext (defined in Sec. 2.3). This ZKAoK is essential to secure our protocol against malicious adversaries. Next, in Sec. 3.2 we explain how parties interactively set up the public parameters of the CL encryption scheme, so that the assumptions underlying the ZKAoK hold. Though – for clarity – we describe this interactive set up as a separate protocol, it can be done in parallel to the $\mathsf{IKeyGen}$ protocol of threshold EC-DSA, thereby only increasing by one the number of rounds of the threshold signing protocol. Finally, in Sec. 3.3 we present our (t, n)-threshold EC-DSA signing protocol, whose security will be demonstrated in Sec. 4.

3.1 ZKAoK ensuring a CL ciphertext is well formed

Consider a prover P having computed an encryption of $a \in \mathbf{Z}/q\mathbf{Z}$ with randomness $r \xleftarrow{\$} [\tilde{A}]$, i.e. $\mathbf{c} := (c_1, c_2)$ with $c_1 := g_q^r,\ c_2 := \mathsf{pk}^r f^a$. We present a zero knowledge argument of knowledge for the following relation:

$$\mathsf{R}_{\mathsf{Enc}} := \{(\mathsf{pk}, \mathbf{c}); (a, r) \mid \mathsf{pk} \in \widehat{G};\ r \in [\tilde{A}C(2^{40}+2)];\ a \in \mathbf{Z}/q\mathbf{Z};\ c_1 = g_q^r \wedge c_2 = \mathsf{pk}^r f^a\}.$$

The interactive protocol is given in Fig. 2. We denote $\mathcal{C}$ the challenge set, and $C := |\mathcal{C}|$. The only constraint on C is that the C-low order assumption holds.

Setup:

1. $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F) \leftarrow \mathsf{Gen}(1^\lambda, q)$.
2. Let $\tilde{A} := \tilde{s} \cdot 2^{40}$, sample $t \xleftarrow{\$} [\tilde{A}]$ and let $g_q := \hat{g}_q^t$.

Prover $((\mathsf{pk}, \mathbf{c}), a \in \mathbf{Z}/q\mathbf{Z}; r \in [\tilde{A}])$		Verifier $((\mathsf{pk}, \mathbf{c}))$
$r_1 \xleftarrow{\$} [2^{40}\tilde{A}C]$		
$r_2 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$		
$t_1 := g_q^{r_1}$		
$t_2 := \mathsf{pk}^{r_1} f^{r_2}$	$\xrightarrow{t_1, t_2}$	
		$k \xleftarrow{\$} \mathcal{C}$
	$\xleftarrow{k}$	
$u_1 := r_1 + kr \in \mathbf{Z}$		
$u_2 := r_2 + ka \in \mathbf{Z}/q\mathbf{Z}$	$\xrightarrow{u_1, u_2}$	Check $u_1 \in [\tilde{A}C(2^{40}+1)]$ and $u_2 \in \mathbf{Z}/q\mathbf{Z}$ and $g_q^{u_1} = t_1 c_1^k$ and $\mathsf{pk}^{u_1} f^{u_2} = t_2 (c_2)^k$

Fig. 2: Zero-knowledge argument of knowledge for $\mathsf{R}_{\mathsf{Enc}}$.

Theorem 2. *If the strong root assumption is $(t'(\lambda), \epsilon_{\mathsf{SR}}(\lambda))$-secure for Gen, and the C-low order assumption is $(t'(\lambda), \epsilon_{\mathsf{LO}}(\lambda))$-secure for Gen, denoting $\epsilon := \max(\epsilon_{\mathsf{SR}}(\lambda), \epsilon_{\mathsf{LO}}(\lambda))$, then the interactive protocol of Fig. 2 is a computationally convincing proof of knowledge for $\mathsf{R}_{\mathsf{Enc}}$ with knowledge error $\kappa(\lambda)$, time bound $t(\lambda)$ and failure probability $\nu(\lambda)$, where $\nu(\lambda) = 8\epsilon$, $t(\lambda) < t'(\lambda)/448$ and $\kappa(\lambda) = \max(4/C, 448t(\lambda)/t'(\lambda))$. If $r \in [\tilde{s} \cdot 2^{40}]$ (it is so when the prover is honest), the protocol is honest verifier statistical zero-knowledge.*

Proof. We prove the properties of soundness, completeness and (honest verifier) zero-knowledge.

*Soundness.*Let us analyse to what extent the protocol of Fig. 2 satisfies the notion of soundness defined in Def. 3, in particular for which knowledge error functions $\kappa()$ is the definition satisfied. Accordingly, let $\kappa()$ be any knowledge error function, such that $\kappa(\lambda) \geq 4/C$ for all λ. We then must define an extractor M. Let a PT prover P^* be given and let view be any view P^* may have after having produced $(\mathsf{pk}, \mathbf{c})$. Now, it can be shown that since there are C different challenges, then if $\epsilon_{\mathsf{view},P} > \kappa(\lambda) \geq 4/C$, standard rewinding techniques allow us to obtain in expected PT a situation where, for given (t_1, t_2), P^* has correctly answered two different values k and k'. We call u_1, u_2 and u_1', u_2' the corresponding answers, so we get:

- $g_q^{u_1} = t_1 \cdot c_1^k$ and $g_q^{u_1'} = t_1 \cdot c_1^{k'}$ s.t. $g_q^{u_1 - u_1'} = c_1^{k-k'}$,
- $\mathsf{pk}^{u_1} f^{u_2} = t_2 \cdot c_2^k$ and $\mathsf{pk}^{u_1'} f^{u_2'} = t_2 \cdot c_2^{k'}$ s.t. $\mathsf{pk}^{u_1 - u_1'} f^{u_2 - u_2'} = c_2^{k-k'}$.

Since $k \neq k'$ and q is prime, with overwhelming probability $(1 - 1/q)$ it holds that $k - k'$ is invertible mod q. In the following we assume this is the case[6].

Let Rewind be a (probabilistic) procedure that creates $k, k', u_1, u_2, u_1', u_2'$ in this way. A concrete algorithm for Rewind is given in [DF02, Appendix A]. It runs in expected time $56/\epsilon_{\mathsf{view},P}$, counting the time to do the protocol once with P^* as one step.

Assume without loss of generality that $k > k'$ and suppose that $(k - k')$ divides $(u_1 - u_1')$ in $\mathbf{Z}$. We denote:

$$\nu_1 := g_q^{(u_1-u_1')/(k-k')} c_1^{-1} \quad \text{and} \quad \nu_2 := \mathsf{pk}^{(u_1-u_1')/(k-k')} f^{(u_2-u_2')/(k-k')} c_2^{-1}.$$

Suppose that $\nu_1 = \nu_2 = 1$. Moreover, $V's$ check on the size of u_1, u_1' implies that $(u_1 - u_1')/(k - k')$ is in the required interval. One can now easily verify that P^* knows $((\mathsf{pk}, \mathbf{c}); ((u_2 - u_2')/(k - k') \mod q, (u_1 - u_1')/(k - k'))) \in \mathsf{R}_{\mathsf{Enc}}$, and from such values $k, k', u_1, u_2, u_1', u_2'$ one can thus extract a witness for the statement.

A set of values $k, k', u_1, u_2, u_1', u_2'$ is said to be bad if $k - k'$ divides $u_1 - u_1'$ but $\nu_1 \neq 1$ or $\nu_2 \neq 1$ or if $k - k'$ does not divide $u_1 - u_1'$. The extractor M simply repeats calling Rewind (for this same $(\mathsf{pk}, \mathbf{c})$) until it gets a set of good values. We will analyse knowledge soundness with this M and the polynomial $p(\lambda)$ from the definition set to the constant of 112. We start with a lemma that gives an exact bound on the security.

Lemma 1. *Let $\mathcal{R}$, (P, V), $\kappa()$, M and $p()$ be as defined above. Given any prover P^*, there exists an algorithm $\mathcal{A}(P^*)$ that solves either the strong root problem for class groups with input $(\widehat{G}, \widehat{G}^q, g_q)$, or the low order problem in $\widehat{G}$ with probability $\mathsf{Adv}_{\kappa,M,p}(P^*, \lambda)/8$, and runs in time $448 \cdot t_{P^*}(k)/\kappa(\lambda)$ where $t_{P^*}(k)$ denotes the running time of P^*.*

Proof. $\mathcal{A}$ does the following: receive $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F, g_q)$ as an input and accordingly set the public parameters for the CL encryption scheme as: $(\tilde{s}, f, \hat{g}_q, g_q, \widehat{G}, F, q)$ as described in Sec. 2.3. Send $(\tilde{s}, f, \hat{g}_q, g_q, \widehat{G}, F, q)$ to P^*, and hope to get a set of bad values. However, if Rewind runs more than $448/\kappa(\lambda)$ times with P^*, we abort and stop. If we obtained a set of bad values, we attempt to compute a root of g_q as described below.

Clearly $\mathcal{A}$ runs in time $448 \cdot t_{P^*}(k)/\kappa(\lambda)$. We now look at its' success probability. Note that the distribution of $(\tilde{s}, f, g_q, \widehat{G}, G, F, G^q)$ that P^* receives here is exactly the same as in a real execution of the protocol. Hence the probability of producing a view for which M fails, is exactly $\mathsf{Adv}_{\kappa,M,p}(P^*, \lambda)$. Note also that given any view view where M fails, it must be the case that the values produced by Rewind are bad with probability of at least $1/2$. If this was not the case, then M could expect to find a witness for $(\mathsf{pk}, \mathbf{c})$ after calling Rewind twice, which takes expected time $\frac{112}{\epsilon_{\mathsf{view},P^*}} \leq \frac{p(\lambda)}{\epsilon_{\mathsf{view},P^*} - \kappa(\lambda)}$ which would contradict the fact M fails on view. So let E be the event that M fails on view and Rewind has returned a set of bad values.

[6] In fact in our application $C < q$, so this holds with probability 1.

Claim. Given that E occurs, we can solve the root problem with probability p^* of at least $1/2$.

To see this, recall that Rewind returns $k, k', u_1, u_2, u_1', u_2'$ s.t. $g_q^{u_1-u_1'} = c_1^{k-k'}$ and $\mathsf{pk}^{u_1-u_1'} f^{u_2-u_2'} = c_2^{k-k'}$.

If $(k-k')$ divides (u_1-u_1') then $\nu_1 \neq 1$ or $\nu_2 \neq 1$ where ν_1 and ν_2 are defined as above. Clearly $\nu_1^{k-k'} = \nu_2^{k-k'} = 1$. And since $k-k' < C$, and one can check that ν_1 and ν_2 are elements of $\widehat{G}$ (by checking that c_1, pk, c_2 are in $\widehat{G}$) this solves the C-low order problem in $\widehat{G}$.

Now let us examine the case where $(k-k')$ does not divide (u_1-u_1'). We denote $d := \gcd(k-k', u_1-u_1')$ and $e := (k-k')/d$; and split in two cases:

Case 1: If $e = 2^m$ for some positive integer m (in which case solving the root problem is easy). The goal will be to show that since P^* does not have control over k, k' this case happens with probability $\leq 1/2$, given that E occurs. Hence the Case 2 where we solve either the root problem of the low order problem happens with large probability, given E. Indeed, observe that for e to be a power of 2, it must hold that $(k-k')$ is a multiple of 2^m, and in particular a multiple of 2. However since k and k' are chosen uniformly at random from C by the honest V, with probability $1/2$, $(k-k')$ is an odd integer.

Case 2: If $e := (k-k')/d$ is not a power of 2. We have $d < |k-k'| < C$. Choose γ and δ s.t. $\gamma(k-k') + \delta(u_1-u_1') = d$. Then $g_q^d = g_q^{\gamma(k-k')+\delta(u_1-u_1')} = (g_q^\gamma c_1^\delta)^{k-k'}$. Now let:

$$\mu := (g_q^\gamma c_1^\delta)^{\frac{(k-k')}{d}} g_q^{-1}.$$

Clearly $\mu^d = 1$, so since $d < C$, if $\mu \neq 1$, we again have a solution to the C-low order problem in $\widehat{G}$. Now suppose that $\mu = 1$. We can rewrite the above as:

$$g_q = (g_q^\gamma c_1^\delta)^{(k-k')/d},$$

which gives a solution for the SRP with input g_q, which is $e = (k-k')/d$ and $X := g_q^\gamma c_1^\delta$, s.t. $g_q = X^e$, $e > 1$ and e is not a power of 2. The claim above now follows.

Summarizing, we therefore have that for every view view where M fails, running Rewind will fail to solve either the strong root problem or the low order problem with probability at most $1 - p^*/2 \leq 3/4$. The expected number of executions of P^* needed to run Rewind is at most $56/\epsilon_{\mathsf{view},P^*} \leq 56/\kappa(\lambda)$. Thus Rewind is allowed to run for at least 8 times its expected running time, and so by the Markov rule it will run for longer with probability at most $1/8$. Since the probability that view is bad in the first place is $\mathsf{Adv}_{\kappa,M,p}(P^*, \lambda)$, the success probability of $\mathcal{A}(P^*)$ is $\mathsf{Adv}_{\kappa,M,p}(P^*, \lambda) \cdot (1 - 1/8 - (1-p^*/2)) \geq \mathsf{Adv}_{\kappa,M,p}(P^*, \lambda)/8$. This finishes the proof.

Completeness. If P knows $r \in [\tilde{A}]$ and $a \in \mathbf{Z}/q\mathbf{Z}$ s.t. $(\mathsf{pk}, \mathbf{c}); (a, r) \in \mathsf{R}_{\mathsf{Enc}}$, and both parties follow the protocol, one has $u_1 \in [\tilde{A}C(2^{40}+1)]$ and $u_2 \in \mathbf{Z}/q\mathbf{Z}$; $\mathsf{pk}^{u_1} f^{u_2} = \mathsf{pk}^{r_1+k\cdot r} f^{r_2+k\cdot a} = \mathsf{pk}^{r_1} f^{r_2} (\mathsf{pk}^r f^a)^k = t_2 c_2^k$; and $g_q^{u_1} = g_q^{r_1+k\cdot r} = t_1 c_1^k$.

Honest verifier zero-knowledge. Given pk, $\mathbf{c} = (c_1, c_2)$ a simulator can sample $k \xleftarrow{\$} [C[$, $u_1 \xleftarrow{\$} [\tilde{A}C(2^{40}+1)]$ and $u_2 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$, compute $t_2 := \mathsf{pk}^{u_1} f^{u_2} c_2^{-k}$ and $t_1 := g_q^{u_1} c_1^{-k}$ such that the transcript $(\mathsf{pk}, \mathbf{c}, t_2, t_1, k, u_1, u_2)$ is indistinguishable from a transcript produced by a real execution of the protocol.

3.2 Interactive set up for the CL encryption scheme

Generating a random generator g_q. In order to use the above ZKAoK it must hold that g_q is a random element of the subgroup $\langle \hat{g}_q \rangle$ where $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F) \leftarrow \mathsf{Gen}(1^\lambda, q)$. Precisely if a malicious prover P^* could break the soundness of the ZKAoK, an adversary S trying to break the SRP, given input a random g_q, should be able to feed this input to P^*, and use P^* to solve it's own challenge. Consequently, as the ZKAoK will be used peer-to-peer by all parties in the threshold EC-DSA protocol, they will collaboratively generate – in the interactive $\mathsf{IKeyGen}$ – the public parameters $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F)$, and a common g_q which is random to each party. We call this interactive sub-protocol ISetup, since it allows parties to collaboratively set up the public parameters for the CL encryption scheme. All parties then use this g_q to compute their public keys and as a basis for the CL encryption scheme. As explained in Sec. 2.3 the generation of $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F)$ is deterministic from a pair of primes $\tilde{q}$ and q, we overload the notation $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F) \leftarrow \mathsf{Gen}(\tilde{q}, q)$ to refer to this deterministic set up. We first define the functionality computed by ISetup, running in two steps.

Definition 9. *For a number of parties n, ISetup consists of the following interactive protocols:*

Step 1 $\langle k; \ldots; k \rangle \to \langle \tilde{q} \rangle$ **or** $\langle \perp \rangle$ *where $\perp$ is the error output, signifying the parties may abort the protocol, and $\tilde{q}$ is a random k bit prime.*

Step 2 $\langle (\tilde{q}, q); \ldots; (\tilde{q}, q) \rangle \to \langle (\tilde{s}, f, \hat{g}_q, \widehat{G}, F, g_q, t_1); \ldots; (\tilde{s}, f, \hat{g}_q, \widehat{G}, F, g_q, t_n) \rangle$ **or** $\langle \perp \rangle$ *where $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F) \leftarrow \mathsf{Gen}(\tilde{q}, q)$, and values $t_1, \ldots, t_n \in [2^{40}\tilde{s}]$ constitute additive shares of t such that $g_q = \hat{g}_q^t$.*

For n parties to collaboratively run ISetup, they proceed as depicted in Fig 3, performing the following steps:
Step 1 — Generation of random public prime $\tilde{q}$ of bit-size k.

1. Each P_i samples a random $r_i \xleftarrow{\$} \{0,1\}^k$, computes $(\mathsf{c}_i, \mathsf{d}_i) \leftarrow \mathsf{Com}(r_i)$ and broadcasts c_i.
2. After receiving $\{\mathsf{c}_j\}_{j \neq i}$, each P_i broadcasts d_i thus revealing r_i.
3. All players compute the common output $\tilde{q} := \mathsf{next\text{-}prime}(\bigoplus_{j=1}^n r_j)$.

Step 2 — Generation of g_q.

1. From $\tilde{q}$, (and the order of the elliptic curve q) all parties can use the deterministic set up of [CL15,CCL+19] which sets a generator $\hat{g}_q$.
2. Next each player P_i performs the following steps:

(a) Sample a random $t_i \xleftarrow{\$} [2^{40}\tilde{s}]$; compute $g_i := \hat{g}_q^{t_i}$; $(\tilde{\mathsf{c}}_i, \tilde{\mathsf{d}}_i) \leftarrow \mathsf{Com}(g_i)$, and broadcast $\tilde{\mathsf{c}}_i$.
(b) Receive $\{\tilde{\mathsf{c}}_j\}_{j \neq i}$. Broadcast $\tilde{\mathsf{d}}_i$ thus revealing g_i.
(c) Perform a ZKPOK of t_i such that $g_i = \hat{g}_q^{t_i}$.[7] If a proof fails, abort.

3. Each party computes $g_q := \prod_{j=1}^{n} g_j = \hat{g}_q^{\sum t_j}$, and has output $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F, g_q, t_i)$.

P_i	$\mathsf{ISetup}(k)$	All players $\{P_j\}_{j \neq i}$
$r_i \xleftarrow{\$} \{0,1\}^k$		
$[\mathsf{c}_i, \mathsf{d}_i] \leftarrow \mathsf{Com}(r_i)$	$\xrightarrow{\mathsf{c}_i}$	
	$\xrightarrow{\mathsf{d}_i}$	$r_i \leftarrow \mathsf{Open}(\mathsf{c}_i, \mathsf{d}_i)$
$\tilde{q} := \mathsf{next\text{-}prime}(\bigoplus_{j=1}^{n} r_j)$		
Compute $\hat{g}_q$ from $q, \tilde{q}$		
$t_i \xleftarrow{\$} [2^{40}\tilde{s}]$ and $g_i \leftarrow \hat{g}_q^{t_i}$		
$(\tilde{\mathsf{c}}_i, \tilde{\mathsf{d}}_i) \leftarrow \mathsf{Com}(g_i)$	$\xrightarrow{\tilde{\mathsf{c}}_i}$	
	$\xrightarrow{\tilde{\mathsf{d}}_i}$	$g_i \leftarrow \mathsf{Open}(\tilde{\mathsf{c}}_i, \tilde{\mathsf{d}}_i)$
$\pi_i := \mathsf{ZKPoK}_{g_i}\{(t_i) : g_i = \hat{g}_q^{t_i}\}$	$\xleftrightarrow{\pi_i}$	if a proof fails abort
$g_q \leftarrow \prod_{j=1}^{n} \hat{g}_q^{t_j} = \prod_{j=1}^{n} g_j$		

Fig. 3: Threshold CL setup used in IKeyGen

Theorem 3 states the security of the interactive protocol ISetup of Fig. 3.

Theorem 3. *If the commitment scheme is non-malleable and equivocal; and the proofs π_i are zero knowledge proofs of knowledge of discrete logarithm in $\langle \hat{g}_q \rangle$, then the protocol of Fig. 3 securely computes* ISetup *with abort, in the presence of a malicious adversary corrupting any $t < n$ parties, with point-to-point channels.*

Proof. We here demonstrate that for each execution of ISetup (*cf.* Fig. 3), which interactively sets the public parameters of the CL framework, our reduction for the strong root problem can program the outputs $\tilde{q}$ and g_q if the reduction controls at least one uncorrupted player.

Indeed consider an adversary S for the SRP for generator Gen. S gets as input a description of $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F)$ output by $\mathsf{Gen}(1^\lambda, q)$, which includes $\tilde{q}$ and the order of the elliptic curve q, and a random element $Y \in \langle \hat{g}_q \rangle$. S must simulate Step 1 so that all players output the same $\tilde{q}$ as S received in the description of G. Next S must simulate Step 2 so that each player P_i outputs $\tilde{s}, f, \hat{g}_q, \widehat{G}, F, g_q = Y$ – of which S must find a root – and some $t_i \in [\tilde{s} \cdot 2^{40}]$. We describe S simulating P_1 against all the other (potentially corrupted) parties, since all parties play symmetric roles, this is without loss of generality.

[7] This can be done as in [CCL+19] (without relying on the strong root assumption).

Simulating step 1 — Generation of $\tilde{q}$.

1. S samples $r_1 \xleftarrow{\$} \{0,1\}^k$, computes $(\mathsf{c}_1, \mathsf{d}_1) := \mathsf{Com}(r_1)$ and broadcasts c_1.
2. S broadcasts d_1, revealing r_1, and receives $\{r_j\}_{j>1}$.
3. S samples r'_1 uniformly at random in $\{0,1\}^k$, subject to the condition $\tilde{q} = \mathsf{next\ prime}(r'_1 \oplus \bigoplus_{j=2}^{n} r_j)$. Then S computes an equivocated decommitment d'_1 which opens to r'_1, rewinds the adversary to 2 and broadcasts d'_1 instead of d_1.
4. All players compute the common output $\tilde{q} := \mathsf{next\ prime}(r'_1 \oplus \bigoplus_{j=2}^{n} r_j)$.

Simulating step 2 — Generation of $Y = g_q$.

1. From $\tilde{q}$ and q all parties use the deterministic set up of [CCL+19] to set generator $\hat{g}_q$.
2. S (simulating P_1) does the following:
   (a) Sample $t_1 \xleftarrow{\$} [2^{40}\tilde{s}]$; compute $g_1 := \hat{g}_q^{t_1}$; $(\tilde{\mathsf{c}}_1, \tilde{\mathsf{d}}_1) = \mathsf{Com}(g_1)$, and broadcast $\tilde{\mathsf{c}}_1$.
   (b) Receive $\{\tilde{\mathsf{c}}_j\}_{j\neq 1}$. Broadcast $\tilde{\mathsf{d}}_1$ thus revealing g_1.
   (c) Perform a ZKPOK for $\pi_1 := \mathsf{ZKPoK}_{g_1}\{(t_1) : g_1 = \hat{g}_q^{t_1}\}$.
   (d) Receive $\{\tilde{\mathsf{c}}_j\}_{j\neq 1}$, recover $g_j \leftarrow \mathsf{Open}(\tilde{\mathsf{c}}_j, \tilde{\mathsf{d}}_j)$ for each j.
   (e) Let $h := \prod_{j=2}^{n} g_j$. Compute $g'_1 := Y \cdot h^{-1}$ and an equivocated decommitment d' which opens to g'_1, rewind the adversary to 2. (b) and broadcast d' instead of $\tilde{\mathsf{d}}_1$. In 2. (b) simulates the ZKPoK.
3. If all the proofs are correct, the protocol goes along with $g_q := g'_1 h = Y$.

Lemma 2. *If the commitment scheme is non-malleable and equivocal; and the proofs π_i are zero knowledge proofs of knowledge then a simulated execution of steps 1 and 2 above is – from the view of (potentially corrupted) parties $P_2, \ldots, P_n$ – indistinguishable from a real execution. Moreover when the simulation – on input (G, g_q), where G is computed deterministically from a prime $\tilde{q}$ – does not abort, all parties output $\tilde{q}$ in step 1, and g_q in step 2.*

Proof. Step 1 The only difference between real and simulated protocols is the way r_1 is computed. In the simulation S does not know r_1, but it chooses a r'_1 such that $\tilde{q} = \mathsf{next\ prime}(r'_1 \oplus \bigoplus_{j=2}^{n} r_j)$. Let $R = \bigoplus_{j=2}^{n} r_j$ and $H_q = \{x \in \{0,1\}^k : \mathsf{prev\text{-}prime}(\tilde{q}) \leq x \oplus R \leq \tilde{q} - 1\}$ be the set of all the elements x such that $\tilde{q} = \mathsf{next\ prime}(x \oplus R)$. Since r_1 belongs to the set H_q, and it has been chosen uniformly at random, as long as r'_1 is chosen uniformly at random in the same set, the real and simulated executions are indistinguishable.
Step 2 The only difference is in point 2.(e), where the simulator computes g'_1 instead of using g_1. Since g_1 and $Y \cdot h^{-1}$ follow the same distribution, real and simulated executions are indistinguishable.

Moreover, we observe that the simulation can fail in three points: in step 1 if someone refuses to decommit after rewinding and in step 2, if some π_i fails or if someone refuses to decommit after rewinding. Since the commitment scheme is

non-malleable and equivocal, in Step 1 the simulator can rewind and equivocate the commitment to r_1, and if there are not aborts, all parties decommit to their correct values. As a consequence, all parties output $\tilde{q}$ at the end of Step 1. In step 2, all parties compute the correct $\hat{g}_q$ using $\tilde{q}$ from the deterministic setup of CL, if not there is an abort caused by the soundness of the proof π_i corresponding to the corrupted P_i. Finally, if no abort has occurred, in step 2, point e), the simulator can equivocate the decommitment to g_1 and all parties decommit to the correct values thanks to the non-malleability of the scheme. If no party refuses to decommit after rewinding, the protocol ends with $g_q = Y$ (and $\tilde{q}$ from step 1). □

Remark 1. The randomness of $\tilde{q}$ is not crucial to the security of the EC-DSA protocol: conversely to RSA prime factors, here $\tilde{q}$ is public. However traditionally, class group based crypto uses random discriminants; we provide a distributed version of the setup of [CL15] in which the prime $\tilde{q}$ is random. In our ISetup algorithm, the output of $\mathsf{next\text{-}prime}$ is biased. To patch this, for the same complexity, parties could jointly generate a seed for a prime pseudo-random generator to generate $\tilde{q}$; such a source of randomness would be sufficient in this context.

3.3 Resulting threshold EC-DSA protocol

We now describe the overall protocol. Participants run on input $(\mathbb{G}, q, P)$ used by the EC-DSA signature scheme. In Fig. 4, and in phases 1, 3, 4, 5 of Fig. 5, all players perform the same operations (on their respective inputs) w.r.t. all other parties, so we only describe the actions of some party P_i. In particular if P_i broadcasts some value v_i, implicitly P_i receives v_j broadcast by P_j for all $j \in [n]$, $j \neq i$. Broadcasts from P_i to all other players are denoted by double arrows, whereas peer-to-peer communications are denoted by single arrows.

On the other hand, Phase 2 of Fig. 5 is performed by all pairs of players $\{(P_i, P_j)\}_{i \neq j}$. Each player will thus perform $(n-1)$ times the set of instructions on the left (performed by P_i on the figure) and $(n-1)$ times those on the right hand side of the figure (performed by P_j).

Key generation. We assume that prior to the interactive key generation protocol $\mathsf{IKeyGen}$, all parties run the ISetup protocol of Sec. 3.2 s.t. they output a common random generator g_q. Each party uses this g_q to generate its' CL encryption key pair, and to verify the ZKAoK in the ISign protocol. Although $\mathsf{IKeyGen}$ and ISetup are here described as two separate protocols, they can be ran in parallel. Consequently, in practice the number of rounds in $\mathsf{IKeyGen}$ increases by 1 broadcast per party if the ZK proofs are made non interactive, and by 2 broadcasts if it is performed interactively between players.
The $\mathsf{IKeyGen}$ protocol (also depicted in Fig 4) proceeds as follows:

1. Each P_i samples a random $u_i \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$; computes $[\mathsf{kgc}_i, \mathsf{kgd}_i] \leftarrow \mathsf{Com}(u_i P)$ and generates a pair of keys $(\mathsf{sk}_i, \mathsf{pk}_i)$ for the CL encryption scheme. Each P_i broadcasts $(\mathsf{pk}_i, \mathsf{kgc}_i)$.

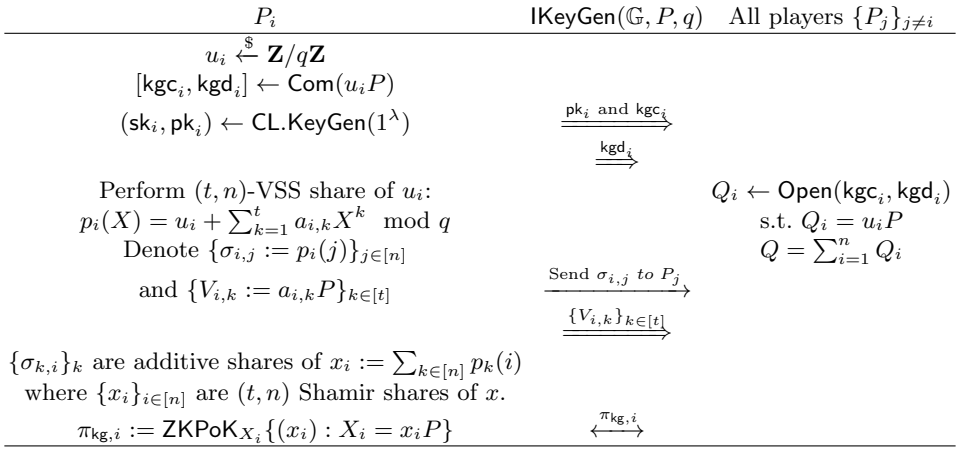

Fig. 4: Threshold Key Generation

2. Each P_i broadcasts kgd_i. Let $Q_i \leftarrow \mathsf{Open}(\mathsf{kgc}_i, \mathsf{kgd}_i)$. Party P_i performs a (t,n) Feldman-VSS of u_i, with Q_i as the free term in the exponent. The EC-DSA public key is set to $Q = \sum_{i=1}^{n} Q_i$. Each player adds the private shares received during the n Feldman VSS protocols. The resulting values x_i are a (t,n) Shamir's secret sharing of the secret signing key x. Observe that all parties know $\{X_i := x_i \cdot P\}_{i\in[n]}$.
3. Each P_i proves in ZK that he knows x_i using Schnorr's protocol [Sch91].

Signing. The signature generation protocol runs on input m and the output of the $\mathsf{IKeyGen}$ protocol of Fig 4. We denote $S \subseteq [n]$ the subset of players which collaborate to sign m. Assuming $|S| = t$ one can convert the (t,n) shares $\{x_i\}_{i\in[n]}$ of x into (t,t) shares $\{w_i\}_{i\in S}$ of x using the appropriate Lagrangian coefficients. Since the $X_i = x_i \cdot P$ and Lagrangian coefficients are public values, all parties can compute $\{W_i := g^{w_i}\}_{i\in S}$. We here describe the steps of the algorithm. A global view of the interactions is also provided in Fig. 5.

Phase 1: Each party P_i samples $k_i, \gamma_i \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ and $r_i \xleftarrow{\$} [\tilde{A}]$ uniformly at random. It computes $c_{k_i} \leftarrow \mathsf{Enc}(\mathsf{pk}_i, k_i; r_i)$, a ZKAoK π_i that the ciphertext is well formed, and $[\mathsf{c}_i, \mathsf{d}_i] \leftarrow \mathsf{Com}(\gamma_i P)$. Each P_i broadcasts $(\mathsf{c}_i, c_{k_i}, \pi_i)$.

Phase 2: *Intuition: denoting* $k := \sum_{i\in S} k_i$ *and* $\gamma := \sum_{i\in S} \gamma_i$ *it holds that* $k\gamma = \sum_{i,j\in S} k_j\gamma_i$ *and* $kx = \sum_{i,j\in S} k_j w_i$*. The aim of Phase 2 is to convert the multiplicative shares* k_j *and* γ_i *of* $(k_j\gamma_i)$ *(resp.* k_j *and* w_i *of* $(k_j w_i)$*) into additive shares* $\alpha_{j,i} + \beta_{j,i} = k_j\gamma_i$ *(resp.* $\mu_{j,i} + \nu_{j,i} = k_j w_i$*). Phase 2 is performed peer-to-peer between each pair* $\{(P_i, P_j)\}_{i\neq j}$*, s.t. at the end of the phase,* P_i *knows* $\{\alpha_{i,j}, \beta_{j,i}, \mu_{i,j}, \nu_{j,i}\}_{j\in S, j\neq i}$.
Each peer-to-peer interaction proceeds as follows:

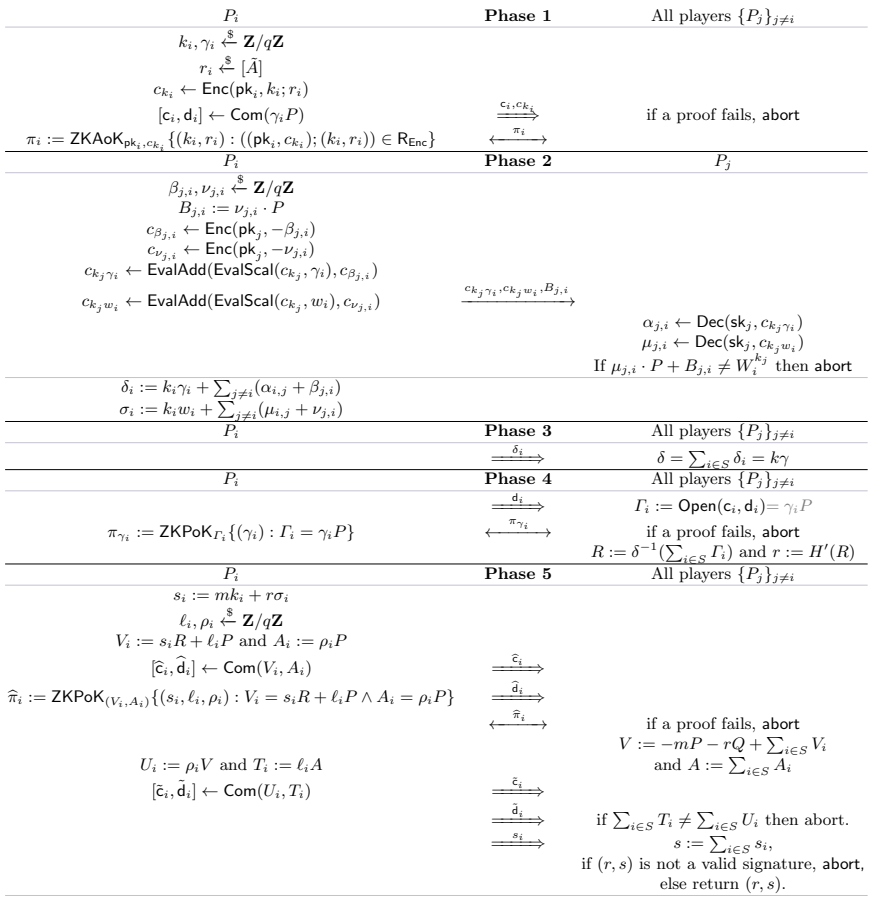

Fig. 5: Threshold signature protocol

(a) P_i samples $\beta_{j,i}, \nu_{j,i} \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$, and computes $B_{j,i} := \nu_{j,i} \cdot P$. It uses the homomorphic properties of the encryption scheme and the ciphertext c_{k_j} broadcast by P_j in Phase 1 to compute $c_{k_j\gamma_i}$ and $c_{k_j w_i}$: encryptions under pk_j of $k_j\gamma_i - \beta_{j,i}$ and $k_j w_i - \nu_{j,i}$ respectively.

(b) P_i sends $(c_{k_j\gamma_i}, c_{k_j w_i}, B_{j,i})$ to P_j, who decrypts both ciphertexts to recover respectively $\alpha_{j,i}$ and $\mu_{j,i}$.

(c) Since W_i is public, P_j verifies that P_i used the same share w_i as that used to compute the public key Q by checking $\mu_{j,i} \cdot P + B_{j,i}$. If the check fails, P_j aborts.

P_i computes $\delta_i := k_i\gamma_i + \sum_{j\neq i}(\alpha_{i,j} + \beta_{j,i})$ and $\sigma_i := k_i w_i + \sum_{j\neq i}(\mu_{i,j} + \nu_{j,i})$.

Phase 3: Each P_i broadcasts δ_i. All players compute $\delta := \sum_{i\in S} \delta_i$.

Phase 4: (a) Each P_i broadcasts d_i which decommits to Γ_i.

(b) Each P_i proves knowledge of γ_i s.t. $\Gamma_i = \gamma_i P$. All players compute $R := \delta^{-1}(\sum_{i\in S} \Gamma_i) = k \cdot P$ and $r := H'(R) \in \mathbf{Z}/q\mathbf{Z}$.

Phase 5: (a) Each P_i computes $s_i = k_i m + \sigma_i r$, samples $\ell_i, \rho_i \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ uniformly at random, computes $V_i := s_i R + \ell_i P$; $A_i := \rho_i P$; and $[\widehat{\mathsf{c}}_i, \widehat{\mathsf{d}}_i] \leftarrow \mathsf{Com}(V_i, A_i)$. Each P_i broadcasts $\widehat{\mathsf{c}}_i$.

(b) Each party P_i decommits by broadcasting $\widehat{\mathsf{d}}_i$ along with a NIZKPoK of (s_i, ℓ_i, ρ_i) s.t. $(V_i = s_i R + \ell_i P) \wedge (A_i = \rho_i P)$. It checks all the proofs it gets from other parties. If a proof fails P_i aborts.

(c) All parties compute $V := -mP - rQ + \sum_{i\in S} V_i$, $A := \sum_{i\in S} A_i$. Each party P_i computes $U_i := \rho_i V$, $T_i := \ell_i A$ and the commitment $[\tilde{\mathsf{c}}_i, \tilde{\mathsf{d}}_i] \leftarrow \mathsf{Com}(U_i, T_i)$. It then broadcasts $\tilde{\mathsf{c}}_i$.

(d) Each P_i decommits to (U_i, T_i) by broadcasting $\tilde{\mathsf{d}}_i$.

(e) All players check $\sum_{i\in S} T_i = \sum_{i\in S} A_i$. If the check fails they abort.

(f) Each P_i broadcasts s_i s.t. all players can compute $s := \sum_{i\in S} s_i$. They check that (r, s) is a valid EC-DSA signature, if so, they output (r, s), otherwise they abort the protocol.

4 Security

The security proof is a reduction to the unforgeability of standard EC-DSA. We demonstrate that if there exists a PPT algorithm $\mathcal{A}$ which breaks the threshold EC-DSA protocol of Fig. 4 and 5, then we can construct a forger $\mathcal{F}$ which uses $\mathcal{A}$ to break the unforgeability of standard EC-DSA. To this end $\mathcal{F}$ must simulate the environment of $\mathcal{A}$, so that $\mathcal{A}$'s view of its interactions with $\mathcal{F}$ are indistinguishable from $\mathcal{A}$'s view in a real execution of the protocol. Precisely, we show that if an adversary $\mathcal{A}$ corrupts $\{P_j\}_{j>1}$, one can construct a forger $\mathcal{F}$ simulating P_1 s.t. the output distribution of $\mathcal{F}$ is indistinguishable from $\mathcal{A}$'s view in an interaction with an honest party P_1 (all players play symmetric roles in the protocol so it is sufficient to provide a simulation for P_1). $\mathcal{F}$ gets as input an EC-DSA public key Q, and has access to a signing oracle for messages of its choice. After this query phase, $\mathcal{F}$ must output a forgery, i.e. a signature σ for a message m of its choice, which it did not receive from the oracle.

4.1 Simulating the key generation protocol

On input a public key $Q := x \cdot P$, the forger $\mathcal{F}$ must set up in its simulation with $\mathcal{A}$ this same public key Q (w/o knowing x). This will allow $\mathcal{F}$ to subsequently simulate interactively signing messages with $\mathcal{A}$, using the output of its' (standard) EC-DSA signing oracle.

The main differences with the proof of [GG18] arise from the fact $\mathcal{F}$ knows it's own decryption key sk_1, but does not extract that of other players. Indeed the encryption scheme we use results from hash proof systems, whose security is statistical, thus the fact $\mathcal{F}$ uses its' secret key does not compromise security, and we can still reduce the security of the protocol to the $\mathsf{ind\text{-}cpa}$-security of the encryption scheme. However as we do not prove knowledge of secret keys associated to public keys in the key generation protocol, $\mathcal{F}$ can not extract the decryption keys of corrupted players. The simulation is described below.

Simulating P_1 in $\mathsf{IKeyGen}$

1. $\mathcal{F}$ receives a public key Q from it's EC-DSA challenger.
2. Repeat the following steps (by rewinding $\mathcal{A}$) until $\mathcal{A}$ sends correct decommitments for $P_2, \ldots, P_n$ on both iterations.
3. $\mathcal{F}$ selects a random value $u_1 \in \mathbf{Z}/q\mathbf{Z}$, computes $[\mathsf{kgc}_1, \mathsf{kgd}_1] \leftarrow \mathsf{Com}(u_1 P)$ and broadcasts kgc_1. $\mathcal{F}$ receives $\{\mathsf{kgc}_j\}_{j\in[n],j\neq 1}$.
4. $\mathcal{F}$ broadcasts kgd_1 and receives $\{\mathsf{kgd}_j\}_{j\in[n],j\neq 1}$. For $i \in [n]$, let $Q_i \leftarrow \mathsf{Open}(\mathsf{kgc}_i, \mathsf{kgd}_i)$ be the revealed commitment value of each party. Each player performs a (t, n) Feldman-VSS of the value Q_i, with Q_i as the free term in the exponent.
5. $\mathcal{F}$ samples a CL encryption key pair $(\mathsf{pk}_1, \mathsf{sk}_1) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$.
6. $\mathcal{F}$ broadcasts pk_1 and receives the public keys $\{\mathsf{pk}_j\}_{j\in[n],j\neq 1}$.
7. $\mathcal{F}$ rewinds $\mathcal{A}$ to the decommitment step and
   - equivocates P_1's commitment to $\widehat{\mathsf{kgd}}$ so that the committed value revealed is now $\widehat{Q}_1 := Q - \sum_{j=2}^{n} Q_j$.
   - simulates the Feldman-VSS with free term $\widehat{Q}_1$.
8. $\mathcal{A}$ will broadcast the decommitments $\{\widehat{\mathsf{kgd}}_j\}_{j\in[n],j\neq 1}$. Let $\{\widehat{Q}_j\}_{j=2\ldots n}$ be the committed value revealed by $\mathcal{A}$ at this point (this could be $\perp$ if $\mathcal{A}$ refuses to decommit).
9. All players compute the public signing key $\widehat{Q} := \sum_{i=1}^{n} \widehat{Q}_i$. If any $Q_i = \perp$ in the previous step, then $\widehat{Q} := \perp$.
10. Each player P_i adds the private shares it received during the n Feldman VSS protocols to obtain x_i (such that the x_i are a (t, n) Shamir's secret sharing of the secret key $x = \sum_i u_i$). Note that due to the free term in the exponent, the values $X_i := x_i \cdot P$ are public.
11. $\mathcal{F}$ simulates the ZKPoK that it knows x_1 corresponding to X_1, and for $j \in [n]$, $j \neq 1$, $\mathcal{F}$ receives from $\mathcal{A}$ a Schnorr ZKPoK of x_j such that $X_j := x_j \cdot P$. $\mathcal{F}$ can extract the values $\{x_j\}_{j\in[n],j\neq 1}$ from these ZKPoK.

4.2 Simulating the signature generation

On input m, $\mathcal{F}$ must simulate the interactive signature protocol from $\mathcal{A}$'s view.

We define $\tilde{k_i} := \mathsf{Dec}(\mathsf{sk}_i, c_{k_i})$, which $\mathcal{F}$ can extract from the proofs Π, and $\tilde{k} := \sum_{i \in S} \tilde{k_i}$. Let $k \in \mathbf{Z}/q\mathbf{Z}$ denote the value s.t. $R := k^{-1} \cdot P$ in Phase 4 of the signing protocol. Notice that if any of the players mess up the computation of R by revealing wrong shares δ_i, we may have $k \neq \tilde{k} \mod q$. As in [GG18], we distinguish two types of executions of the protocol: an execution where $\tilde{k} = k \mod q$ is said to be *semi-correct*, whereas an execution where $\tilde{k} \neq k \mod q$ is *non semi-correct*. Both executions will be simulated differently. At the end of Phase 4, when both simulations diverge, $\mathcal{F}$ knows k and $\tilde{k}$, so it can detect if it is in a semi-correct execution or not and chose how to simulate P_1.

We point out that $\mathcal{F}$ does not know the secret share w_1 of x associated with P_1, but it knows the shares $\{w_j\}_{j \in S, j \neq 1}$ of all the other players. Indeed $\mathcal{F}$ can compute these from the values $\{x_j\}_{j \in [n], j \neq 1}$ extracted during key generation. It also knows $W_1 = w_1 \cdot P$ from the key generation protocol. Moreover $\mathcal{F}$ knows the encryption keys $\{\mathsf{pk}_j\}_{j \in S}$ of all players, and it's own decryption key sk_1.

In the following simulation $\mathcal{F}$ aborts whenever $\mathcal{A}$ refuses to decommit any of the committed values, fails a ZK proof, or if the signature (r, s) does not verify.

Simulating P_1 in ISign

Phase 1: As in a real execution, $\mathcal{F}$ samples $k_1, \gamma_1 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ and $r_1 \xleftarrow{\$} [\tilde{A}]$ uniformly at random. It computes $c_{k_1} \leftarrow \mathsf{Enc}(\mathsf{pk}_1, k_1; r_1)$, the associated ZKAoK Π_1, and $[\mathsf{c}_1, \mathsf{d}_1] \leftarrow \mathsf{Com}(\gamma_1 P)$. It broadcasts $(\mathsf{c}_1, c_{k_1}, \Pi_1)$ before receiving $\{\mathsf{c}_j, c_{k_j}, \Pi_j\}_{j \in S, j \neq 1}$ from $\mathcal{A}$. $\mathcal{F}$ checks the proofs are valid and extracts the encrypted values $\{k_j\}_{j \in S, j \neq 1}$ from which it computes $\tilde{k} := \sum_{i \in S} k_i$.

Phase 2: (a) For $j \in S, j \neq 1$, $\mathcal{F}$ computes $\beta_{j,1}$, $c_{k_j \gamma_1}$ as in a real execution of the protocol, however since it only knows $W_1 = w_1 P$ (but not w_1), it samples a random $\mu_{j,1} \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ and sets $c_{k_j w_1} \leftarrow \mathsf{Enc}(\mathsf{pk}_j, \mu_{j,1})$, and $B_{j,1} := k_j \cdot W_1 - \mu_{j,1} \cdot P$. $\mathcal{F}$ then sends $(c_{k_j \gamma_1}, c_{k_j w_1}, B_{j,1})$ to P_j.

(b) When it receives $(c_{k_1 \gamma_i}, c_{k_1 w_j}, B_{1,j})$ from P_j, it decrypts as in a real execution of the protocol to obtain $\alpha_{1,j}$ and $\mu_{1,j}$

(c) $\mathcal{F}$ verifies that $\mu_{1,j} P + B_{1,j} = k_1 W_j$. If so, since $\mathcal{F}$ also knows k_1 and w_j, it computes $\nu_{1,j} = k_1 w_j - \mu_{1,j} \mod q$

$\mathcal{F}$ computes $\delta_1 := k_1 \gamma_1 + \sum_{k \neq 1} \alpha_{1,k} + \sum_{k \neq 1} \beta_{k,1}$. However $\mathcal{F}$ cannot compute σ_1 since it does not know w_1, but it can compute

$$\begin{aligned} \sum_{i>1} \sigma_i &= \sum_{i>1} (k_i w_i + \sum_{j \neq i} \mu_{i,j} + \nu_{j,i}) = \sum_{i>1} \sum_{j \neq i} (\mu_{i,j} + \nu_{j,i}) + \sum_{i>1} k_i w_i \\ &= \sum_{i>1} (\mu_{i,1} + \nu_{1,i}) + \sum_{i>1; j>1} k_i w_j \end{aligned}$$

since it knows all the values $\{k_j\}_{j \in S}$, $\{w_j\}_{j \in S, j \neq 1}$, it chooses the random values $\mu_{i,1}$ and it can compute all of the shares $\nu_{1,j} = k_1 w_j - \mu_{1,j} \mod q$.

Phase 3: $\mathcal{F}$ broadcasts δ_1 and receives all the $\{\delta_j\}_{j \in S, j \neq 1}$ from $\mathcal{A}$. Let $\delta := \sum_{i \in S} \delta_i$.

Phase 4: (a) $\mathcal{F}$ broadcasts d_1 which decommits to Γ_1, and $\mathcal{A}$ reveals $\{\mathsf{d}_j\}_{j\in S, j\neq 1}$ which decommit to $\{\Gamma_j\}_{j\in S, j>1}$.

(b) $\mathcal{F}$ proves knowledge of γ_1 s.t. $\Gamma_1 = \gamma_1 P$, and for $j \in S, j \neq 1$, receives the PoK of γ_j s.t. $\Gamma_j = \gamma_j P$. $\mathcal{F}$ extracts $\{\gamma_j\}_{j\in S, j\neq 1}$ from which it computes $\gamma := \sum_{i\in S} \gamma_i \mod q$ and $k := \delta \cdot \gamma^{-1} \mod q$.

(c) If $k = \tilde{k} \mod q$ (semi-correct execution), $\mathcal{F}$ proceeds as follows:
- $\mathcal{F}$ requests a signature (r, s) for m from its EC-DSA signing oracle.
- $\mathcal{F}$ computes $R := s^{-1}(m \cdot P + r \cdot Q) \in \mathbb{G}$ (note that $r = H'(R) \in \mathbf{Z}/q\mathbf{Z}$).
- $\mathcal{F}$ rewinds $\mathcal{A}$ to the decommitment step at Phase 4. (a) and equivocates P_1's commitment to open to $\widehat{\Gamma}_1 := \delta \cdot R - \sum_{i>1} \Gamma_i$. It also simulates the proof of knowledge of $\widehat{\gamma}_1$ s.t. $\widehat{\Gamma}_1 = \widehat{\gamma}_1 P$. Note that $\delta^{-1}(\widehat{\Gamma}_1 + \sum_{i>1} \Gamma_i) = R$.

Phase 5: Now $\mathcal{F}$ knows $\sum_{j\in S, j\neq 1} s_j$ held by $\mathcal{A}$ since $s_j = k_j m + \sigma_j r$.
- $\mathcal{F}$ computes s_1 held by P_1 as $s_1 := s - \sum_{j\in S, j\neq 1} s_j$.
- $\mathcal{F}$ continues the steps of Phase 5 as in a real execution.

(d) Else $k \neq \tilde{k} \mod q$ (non-semi-correct), and $\mathcal{F}$ proceeds as follows:
- $\mathcal{F}$ computes $R := \delta^{-1}(\sum_{i\in S} \Gamma_i) = k \cdot P$ and $r := H'(R) \in \mathbf{Z}/q\mathbf{Z}$.
- Phase 5: $\mathcal{F}$ does the following
  - sample a random $\tilde{s}_1 \xleftarrow{\$} Zq$.
  - sample $\ell_1, \rho_1 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$, compute $V_1 := s_1 R + \ell_1 P$; $A_1 := \rho_1 P$; $[\widehat{\mathsf{c}}_1, \widehat{\mathsf{d}}_1] \leftarrow \mathsf{Com}(V_1, A_1)$ and send $\widehat{\mathsf{c}}_1$ to $\mathcal{A}$.
  - receive $\{\widehat{\mathsf{c}}_j\}_{j\neq 1}$ and decommit by broadcasting $\widehat{\mathsf{d}}_1$. Prove knowledge of (s_1, ℓ_1, ρ_1) s.t. $(V_1 = s_1 R + \ell_1 P) \wedge (A_1 = \rho_1 P)$.
  - For $j \in S$, $j \neq 1$, $\mathcal{F}$ receive $\widehat{\mathsf{d}}_j$ and the ZKPoK of (s_j, ℓ_j, ρ_j) s.t. $V_j = s_j R + \ell_j P \wedge A_j = \rho_j P$.
  - Compute $V := -mP - rQ + \sum_{i\in S} V_i$, $A := \sum_{i\in S} A_1$, $T_1 := \ell_1 A$ and sample a random $U_1 \xleftarrow{\$} \mathbb{G}$.
  - Compute $[\tilde{\mathsf{c}}_1, \tilde{\mathsf{d}}_1] \leftarrow \mathsf{Com}(U_1, T_1)$ and send $\tilde{\mathsf{c}}_1$ to $\mathcal{A}$. Upon receiving $\{\tilde{\mathsf{c}}_j\}_{j\neq 1}$ from $\mathcal{A}$, broadcast $\tilde{\mathsf{d}}_1$ and receive the $\{\tilde{\mathsf{d}}_j\}_{j\neq 1}$.
  - Now since $\sum_{i\in S} T_1 \neq \sum_{i\in S} U_1$ both $\mathcal{A}$ and $\mathcal{F}$ abort.

4.3 The simulation of a semi-correct execution

Lemma 3. *Assuming the strong root assumption and the C-low order assumption hold for Gen; the CL encryption scheme is $\mathsf{ind\text{-}cpa}$-secure; and the commitment scheme is non-malleable and equivocable; then on input m the simulation either outputs a valid signature (r, s) or aborts, and is computationally indistinguishable from a semi-correct real execution.*

Proof. The differences between the real and simulated views are the following:

1. $\mathcal{F}$ does not know w_1, so it cannot compute $c_{k_j w_1}$ as in a real execution of the protocol. However under the strong root and C-low order assumption in

$\widehat{G}$, $\mathcal{F}$ can extract k_j from the proofs in Phase 1. It then samples a random $\mu_{j,1} \in \mathbf{Z}/q\mathbf{Z}$, computes $B_{j,1} := k_j \cdot W_1 - \mu_{j,1} \cdot P$, and $c_{k_j w_1} \leftarrow \mathsf{Enc}(\mathsf{pk}_j, \mu_{j,1})$. The resulting view of $\mathcal{A}$ is indistinguishable from an honestly generated one since $\mu_{j,1}$ is uniformly distributed in $\mathbf{Z}/q\mathbf{Z}$, both in real and simulated executions; c_{k_j} was proven to be a valid ciphertext, so ciphertexts computed using homomorphic operations over c_{k_j} and and fresh ciphertexts computed with pk_j follow identical distributions from $\mathcal{A}$'s view. And finally $B_{j,1}$ follows a uniform distribution in $\mathbb{G}$ both in real and simulated executions, and passes the check $B_{j,1} + \mu_{j,1} \cdot P = k_j \cdot W_1$ performed by $\mathcal{A}$.

2. $\mathcal{F}$ computes $\widehat{\Gamma}_1 := \delta \cdot R - \sum_{i>1} \Gamma_i$, and equivocates its commitment c_1 s.t. d_1 decommits to $\widehat{\Gamma}_1$. Let us denote $\widehat{\gamma}_1 \in \mathbf{Z}/q\mathbf{Z}$ the value s.t. $\widehat{\Gamma}_1 = \widehat{\gamma}_1 P$, where $\widehat{\gamma}_1$ is unknown to $\mathcal{F}$, but the forger can simulate the ZKPoK of $\widehat{\gamma}_1$.
   Let us further denote $\widehat{k} \in \mathbf{Z}/q\mathbf{Z}$ the randomness (unknown to $\mathcal{F}$) used by its' signing oracle to produce (r, s). It holds that $\delta = \widehat{k}(\widehat{\gamma}_1 + \sum_{j \in S, j>1} \gamma_j)$. Finally, let us denote $\widehat{k}_1 := \widehat{k} - \sum_{j \in S, j>1} k_j$.
   Since δ was made public in Phase 3, by decommiting to $\widehat{\Gamma}_1 = \widehat{\gamma}_1 P$ instead of $\Gamma_1 = \gamma_1 P$, $\mathcal{F}$ is implicitly using $\widehat{k}_1 \neq k_1$, even though $\mathcal{A}$ received an encryption of k_1 in Phase 2. However, if $\mathcal{A}$ could tell apart a real and simulated execution based on this difference, one could use $\mathcal{A}$ to break the indistinguishabilty of the encryption scheme. So, under the assumption the CL encryption scheme is $\mathsf{ind\text{-}cpa}$-secure, this change is unnoticeable to $\mathcal{A}$.
3. $\mathcal{F}$ does not know σ_1, and thus cannot compute s_1 as in a real execution. Instead it computes $s_1 = s - \sum_{j \in S, j \neq 1} s_j = s - \sum_{j \in S, j \neq 1} (k_j m + \sigma_j r)$ where (implicitly) $s = \widehat{k}(m + rx)$. So $s_1 = \widehat{k}_1 m + r(\widehat{k}x - \sum_{j \in S, j \neq 1} \sigma_j)$, and $\mathcal{F}$ is implicitly setting $\widehat{\sigma}_1 := \widehat{k}x - \sum_{j \in S, j \neq 1} \sigma_j$ s.t. $\widehat{k}x = \widehat{\sigma}_1 + \sum_{j \in S, j \neq 1} \sigma_j$.
   We note that, since the real execution is semi correct, the correct shares of k for the adversary are the k_i that the simulator knows and $R = \widehat{k}P = (\widehat{k_1} + \sum_{j \in S, j \neq 1} k_j)$. Therefore the value s_1 computed by $\mathcal{F}$ is consistent with a correct share for P_1 for a valid signature (r, s), which makes Phase 5 indistinguishable from the real execution to the adversary.
   In particular, observe that if none of the parties aborted during Phase 2, the output shares are correct. So if $\mathcal{A}$ here uses the values $\{\sigma_j\}_{j \in S, j>1}$ as computed in a real execution of the protocol, it expects the signature generation protocol to output a valid signature. And indeed with $\mathcal{F}$'s choice of $\widehat{\sigma}_1$ and $\widehat{k}_1$, the protocol will terminate, outputting the valid signature (r, s) it received from its signing oracle. Conversely, if $\mathcal{A}$ attempts to cheat in Phase 5 by using a different set of σ_j's than those prescribed by the protocol, the check $\sum_{i \in S} T_i = \sum_{i \in S} U_i$ will fail, and all parties abort, as in a real execution of the protocol. ■

4.4 Non semi-correct executions

Lemma 4. *Assuming the strong root assumption and the C-low order assumption hold for Gen; the DDH assumption holds in $\mathbb{G}$; and the commitment scheme*

is non-malleable and equivocable; then the simulation is computationally indistinguishable from a non-semi-correct real execution.

Proof. We construct three games between the simulator $\mathcal{F}$ (running P_1) and the adversary $\mathcal{A}$ (running all other players). In G_0, $\mathcal{F}$ runs the real protocol. The only change between G_0 and G_1 is that in G_1, $\mathcal{F}$ chooses U_1 as a random group element. In G_2 the simulator $\mathcal{F}$ runs the simulation described in Sec. 4.2.

Indistinguishability of G_0 and G_1. We prove that if there exists an adversary $\mathcal{A}_0$ distinguishing games G_0 and G_1, $\mathcal{A}_0$ can be used to break the DDH assumption in $\widehat{G}$. Let $\tilde{A} = a \cdot P$, $\tilde{B} = b \cdot P$, $\tilde{C} = c \cdot P$ be the DDH challenge where $c = ab$ or c is random in $\mathbb{Z}_q$. The DDH distinguisher $\mathcal{F}_0$ runs $\mathcal{A}_0$, simulating the key generation phase s.t. $Q = \tilde{B}$. It does so by rewinding $\mathcal{A}_0$ in step 7 of the $\mathsf{IKeyGen}$ simulation and changing the decommitment of P_1 to $Q_1 := \tilde{B} - \sum_{j\in[n],j\neq 1} Q_j$. $\mathcal{F}_0$ also extracts the values $\{x_j\}_{j\in[n],j\neq 1}$ chosen by $\mathcal{A}_0$ from the ZKPoK of step 11 of the $\mathsf{IKeyGen}$ simulation. Note that at this point $Q = \tilde{B}$ and $\mathcal{F}_0$ knows x_i and the decryption key sk_1 matching pk_1, but not b and therefore not x_1.

Next $\mathcal{F}_0$ runs the signature generation protocol for a non-semi-correct execution. Recall that $S \subseteq [n]$ denotes the subset of players collaborating in ISign. Denoting $t := |S|$, the (t, n) shares $\{x_i\}_{i\in' n]}$ are converted into (t, t) shares $\{w_i\}_{i\in S}$ as per the protocol. Thus $b = \sum_{i\in S} w_i$ where $\mathcal{F}_0$ knows $\{w_j\}_{j\in S,j\neq 1}$ but not w_1. We denote $w_A := \sum_{j\in S,j\neq 1} w_j$ (which is known to $\mathcal{F}_0$) s.t. $w_1 = b - w_A$. $\mathcal{F}_0$ runs the protocol normally for Phases $1, 2, 3, 4$. It extracts the values $\{\gamma_j\}_{j\in S,j\neq 1}$ from the proof of knowledge in Phase 4, and knows γ_1 since it ran P_1 normally. Therefore $\mathcal{F}_0$ knows k such that $R = k^{-1} \cdot P$ since $k = (\sum_i \gamma_i)^{-1}\delta \mod q$. It also knows k_1 (chosen normally according to the protocol) and $\{k_j\}_{j\in S,j\neq 1}$ which it can extract from the proofs in Phase 1.

Before moving to the simulation of Phase 5, let's look at Phase 2 of the protocol for the computation of the shares σ_i. We note that since $\mathcal{F}_0$ knows sk_1 it also knows all the shares $\mu_{1,j}$ since it can decrypt the ciphertext $c_{k_1 w_j}$ it receives from P_j. However $\mathcal{F}_0$ does not know w_1 therefore it sends the encryption of a random $\mu_{j,1}$ to P_j and sets (implicitly) $\nu_{j,1} = k_j w_1 - \mu_{j,1}$. At the end the share σ_1 held by P_1 is

$$\sigma_1 = k_1 w_1 + \sum_{j\in S,j\neq 1} (\mu_{1,j} + \nu_{j,1}) = \tilde{k} w_1 + \sum_{j\in S,j\neq 1} (\mu_{1,j} - \mu_{j,1}) \text{ where } \tilde{k} = \sum_{i\in S} k_i.$$

Recall that since this is a non-semi-correct execution $\tilde{k} \neq k$ where $R = k^{-1} \cdot P$. Since $w_1 = b - w_A$ we have $\sigma_1 = \tilde{k} b + \mu_1$ where $\mu_1 = \sum_{j\in S,j\neq 1} (\mu_{1,j} - \mu_{j,1}) - \tilde{k} w_A$ with $\mu_1, \tilde{k}$ known to $\mathcal{F}_0$. This allows $\mathcal{F}_0$ to compute the correct value $\sigma_1 \cdot P = \tilde{k}\tilde{B} + \mu_1 \cdot P$ and therefore the correct value of $s_1 \cdot R$ as:

$$\begin{aligned} s_1 \cdot R &= (k_1 m + r\sigma_1) \cdot R = k^{-1}(k_1 m + r\sigma_1) \cdot P \\ &= k^{-1}(k_1 m + r\mu_1) \cdot P + k^{-1}(\tilde{k} r) \cdot \tilde{B} = \hat{\mu}_1 \cdot P + \hat{\beta}_1 \cdot \tilde{B} \end{aligned}$$

where $\hat{\mu}_1 = k^{-1}(k_1 m + r\mu_1)$ and $\hat{\beta}_1 = k^{-1}\tilde{k} r$ are known to $\mathcal{F}_0$.

In the simulation of Phase 5, $\mathcal{F}_0$ selects a random ℓ_1 and sets $V_1 := s_1 \cdot R + \ell_1 \cdot P$, $A_1 = \rho_1 \cdot P = \tilde{A} = a \cdot P$. It simulates the ZK proof (since it does not know ρ_1 or s_1). It extracts s_i, ℓ_i, ρ_i from $\mathcal{A}_0$'s proofs s.t. $V_i = s_i \cdot R + \ell_i \cdot P = k^{-1} s_i \cdot P + \ell_i \cdot P$ and $A_i = \rho_i \cdot P$. Let $s_A = \sum_{j \in S, j \neq 1} k^{-1} s_j$. Note that, substituting the above relations (and setting $\ell = \sum_{i \in S} \ell_i$), we have: $V = -m \cdot P - r \cdot Q + \sum_{i \in S} V_i = \ell \cdot P + s_1 \cdot R + (s_A - m) \cdot P - r \cdot Q$. Moreover $Q = \tilde{B}$ so $-r \cdot Q = -r \cdot \tilde{B}$, and:

$$V = \ell \cdot P + \hat{\mu}_1 \cdot P + \hat{\beta}_1 \cdot \tilde{B} + (s_A - m) \cdot P - r \cdot \tilde{B} = (\ell + \theta) \cdot P + \kappa \cdot \tilde{B}$$

where $\mathcal{F}_0$ knows $\theta = \hat{\mu}_1 + s_A - m$ and $\kappa = \hat{\beta}_1 - r$. Note that for executions that are not semi-correct $\kappa \neq 0$.

Next $\mathcal{F}_0$ computes $T_1 := \ell_1 \cdot A$ (correctly), but computes U_1 as $U_1 := (\ell + \theta) \cdot \tilde{A} + \kappa \cdot \tilde{C}$, using this U_1 it continues as per the real protocol and aborts on the check $\sum_{i \in S} T_i = \sum_{i \in S} U_i$.

Observe that when $\tilde{C} = ab \cdot P$, by our choice of $a = \rho_1$ and $b = x$, we have that $U_1 = (\ell + \theta)\rho_1 \cdot P + \kappa \cdot \rho_1 \tilde{B} = \rho_1 \cdot V$ as in Game G_0. However when $\tilde{C}$ is a random group element, U_1 is uniformly distributed as in G_1. Therefore under the DDH assumption G_0 and G_1 are indistinguishable.

Indistinguishability of G_1 and G_2. In G_2, $\mathcal{F}$ broadcasts a random $\tilde{V}_1 = \tilde{s}_1 \cdot R + \ell_1 \cdot P$. This is indistinguishable from the correct $V_1 = s_1 \cdot R + \ell_1 \cdot P$ thanks to the mask $\ell_1 \cdot P$ which (under the DDH assumption) is computationally indistinguishable from a random value, since the adversary only knows A_1. To be precise, let $\tilde{A} = (a - \delta) \cdot P$, $\tilde{B} = b \cdot P$ and $\tilde{C} = ab \cdot P$ be the DDH challenge where δ is either 0 or random in $\mathbb{Z}_q$. The simulator proceeds as in G_0 (i.e. the regular protocol) until Phase 5. In Phase 5 $\mathcal{F}_0$ broadcasts $V_1 = \tilde{s}_1 \cdot R + \tilde{A}$ and $A_1 = \tilde{B}$. It simulates the ZKPoK (it does not know ℓ_1 or ρ_1), and extracts s_i, ℓ_i, ρ_i from the adversary s.t. $V_i = s_i \cdot R + \ell_i \cdot P = k^{-1} s_i \cdot P + \ell_i \cdot P$ and $A_i = \rho_i \cdot P$.

Next $\mathcal{F}_0$ samples a random U_1 and sets $T_1 := \tilde{C} + \sum_{j \in S, j \neq 1} \rho_j \cdot \tilde{A}$ before aborting. Note that when $\tilde{A} = a \cdot P$, we implicitly set $a = \ell_1$ and $b = \rho_1$ and have $V_1 = s_1 \cdot R + \ell_1 \cdot P$ and $T_1 = \ell_1 \cdot A$ as in Game G_1. However when $\tilde{A} = a \cdot P - \delta \cdot P$ with a random δ, then this is equivalent to having $V_1 = \tilde{s}_1 \cdot R + \ell_1 \cdot P$ and $T_1 = \ell_1 \cdot A$ with a randomly distributed $\tilde{s}_1$ as in Game G_2. Therefore under the DDH assumption G_1 and G_2 are indistinguishable.

4.5 Concluding the proof

As mentioned at the beginning of Sec. 4.2 the forger $\mathcal{F}$ simulating $\mathcal{A}$'s environment can detect whether we are in a semi-correct-execution or not, i.e. whether $\mathcal{A}$ decides to be malicious and terminate the protocol with an invalid signature. Consequently $\mathcal{F}$ always knows how to simulate $\mathcal{A}$'s view and all simulations are indistinguishable of real executions of the protocol. Moreover if $\mathcal{A}$, having corrupted up to t parties in the threshold EC-DSA protocol, outputs a forgery, since $\mathcal{F}$ set up with $\mathcal{A}$ the same public key Q as it received from its' EC-DSA challenger, $\mathcal{F}$ can use this signature as its own forgery, thus breaking the existential unforgeability of standard EC-DSA.

Denoting $\mathsf{Adv}^{\mathsf{tu\text{-}cma}}_{\Pi,\mathcal{A}}$, $\mathcal{A}$'s advantage in breaking the existential unforgeability of our threshold protocol, and $\mathsf{Adv}^{\mathsf{eu\text{-}cma}}_{\mathsf{ecdsa},\mathcal{F}}$ the forger $\mathcal{F}$'s advantage in breaking the existential unforgeability of standard EC-DSA, from Lemmas 3 and 4 it holds that if the DDH assumption holds in $\mathbb{G}$; the strong root assumption and the C-low order assumption hold for Gen; the CL encryption scheme is ind-cpa-secure; and the commitment scheme is non-malleable and equivocable then: $|\mathsf{Adv}^{\mathsf{eu\text{-}cma}}_{\mathsf{ecdsa},\mathcal{F}} - \mathsf{Adv}^{\mathsf{tu\text{-}cma}}_{\Pi,\mathcal{A}}| \leq \mathsf{negl}(\lambda)$. Under the security of the EC-DSA signature scheme, $\mathsf{Adv}^{\mathsf{eu\text{-}cma}}_{\mathsf{ecdsa},\mathcal{F}}$ must be negligible, which implies that $\mathsf{Adv}^{\mathsf{tu\text{-}cma}}_{\Pi,\mathcal{A}}$ should too, thus contradicting the assumption that $\mathcal{A}$ has non-negligible advantage of forging a signature for our protocol. We can thus state the following theorem, which captures the security of the protocol.

Theorem 4. *Assuming standard EC-DSA is an existentially unforgeable signature scheme; the* DDH *assumption holds in* $\mathbb{G}$*; the strong root assumption and the C-low order assumption hold for Gen; the* CL *encryption scheme is* ind-cpa*-secure; and the commitment scheme is non-malleable and equivocable, then the (t, n)-threshold EC-DSA protocol of Fig. 4 and 5 is an existentially unforgeable threshold signature scheme.*

5 Further improvements

5.1 An improved ZKPoK which kills low order elements.

We here provide a proof of knowledge of discrete logarithm in a group of unknown order. Traditionally, if one wants to perform such a proof, the challenge set must be binary, which implies expensive protocols as the proof must be repeated many times to achieve a satisfying (non computational) soundness error. Here using what we call the *lowest common multiple* trick, we are able to significantly increase the challenge set, and thereby reduce the number of repetitions required of the proof. We first present the resulting proof, before providing two applications: one for the CL.ISetup protocol of Sec. 3.2, and another for the two party EC-DSA protocol of [CCL+19]. Throughout this subsection we denote $y := \mathrm{lcm}(1, 2, 3, \ldots, 2^{10})$.

The lowest common multiple trick. For a given statement h, the proof does not actually prove knowledge of the Dlog of h, but rather of h^y. Precisely, the protocol of Fig. 6 is a zero knowledge proof of knowledge for the following relation:

$$\mathsf{R}_{\mathsf{lcm-DL}} := \{(h, g_q); z \;\mid\; h^y = g_q^z\}.$$

Correctness. If $h = g_q^x$, then $g_q^u = g_q^{r+kx} = g_q^r \cdot (g_q^x)^k = t \cdot h^k$ and V accepts.
Special soundness. Suppose that for a committed value t, prover P^* can answer correctly for two different challenges k_1 and k_2. We call u_1 and u_2 the two answers. Let $k := k_1 - k_2$ and $u := u_1 - u_2$, then since $g_q^{u_1} = t \cdot h^{k_1}$ and $g_q^{u_2} = t \cdot h^{k_2}$, it holds that $g_q^u = h^k$. By the choice of the challenge set, y/k is an integer and so $(g_q^u)^{y/k} = (h^k)^{y/k} = h^y$. Denoting $z := uy/k$, P^* can compute

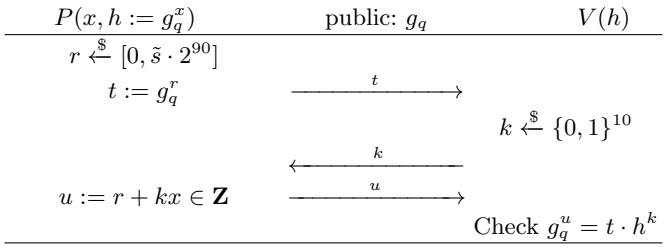

Fig. 6: ZKPoK of z s.t. $h^y = g_q^z$ where $y = \mathrm{lcm}(1, 2, 3, \ldots, 2^{10})$

z such that $g_q^z = h^y$, so if P can convince V for two different challenge values, then P^* can compute a z satisfying the relation.

Zero knowledge. Given h a simulator can sample $k \xleftarrow{\$} \{0,1\}^{10}$ and $u \xleftarrow{\$} [0, \tilde{s} \cdot (2^{90} + k)]$, compute $t := g_q^u \cdot h^{-k}$, such that distribution of the resulting transcript (h, t, k, u) is statistically close to those produced by a real execution of the protocol (this holds since an honest prover samples x from $[\tilde{s} \cdot 2^{40}]$, the challenge space is of size 2^{10} and r is sampled from a set of size $\tilde{s} \cdot 2^{90}$, which thus statistically hides kx).

Application to the CL *interactive set up.* In the ISetup protocol of Sec. 3.2, in Step 2. 2. (c) each P_i computes $\pi_i := \mathsf{ZKPoK}_{g_i}\{(t_i) : g_i = \widehat{g}_q^{t_i}\}$. In fact it suffices for them to compute $\mathsf{ZKPoK}_{g_i}\{(z_i) : g_i^y = \widehat{g}_q^{z_i}\}$, where $y := \mathrm{lcm}(1, 2, 3, \ldots, 2^{10})$ using the lcm trick. Then in Step 2. 3. all players compute $g_q := (\prod_{j=1}^n g_j)^y$. The resulting g_q has the required properties to be plugged into the IKeyGen protocol. We use this modified interactive set up for our efficiency comparisons of Sec. 6.

Application to the [CCL+19] interactive key generation. Castagnos et al. recently put forth a generic two party EC-DSA protocol from hash proof systems [CCL+19]. They rely on a ZKPoK for the following relation:

$$\mathsf{R}_{\mathsf{CL-DL}} := \{(\mathsf{pk}, (c_1, c_2), Q); (x, r) \;\mid\; c_1 = g_q^r \wedge c_2 = f^x \mathsf{pk}^r \wedge Q = xP\}.$$

The interactive proof they provide uses binary challenges, consequently in order to achieve a satisfying soundness error of $2^{-\lambda}$, the proof must be repeated λ times. Using the lcm trick one can divide by 10 this number of rounds, though we obtain a ZKPoK for the following relation:

$$\mathsf{R}_{\mathsf{CL-lcm}} := \{(\mathsf{pk}, (c_1, c_2), Q); (x, z) \;\mid\; c_1^y = g_q^z \wedge c_2^y = f^{x \cdot y} \mathsf{pk}^z \wedge Q = xP\}.$$

In their protocol this ZKPoK is computed by Alice, who sends this proof to Bob s.t. he is convinced her ciphertext $\mathbf{c} = (c_1, c_2)$ is well formed. Bob then performs some homomorphic operations on $\mathbf{c}$ and sends the result back to Alice. Now since with the proof based on the lcm trick, Bob is only convinced that $\mathbf{c}^y$ is a valid ciphertext, Bob raises $\mathbf{c}$ to the power y before performing his homomorphic

operations[8]. When Alice decrypts she multiplies the decrypted value by y^{-1} mod q (this last step is much more efficient than Bob's exponentiation).

Remark 2. The size of the challenge set $\mathcal{C}$ from which k is sampled determines the number of times the protocol needs to be repeated in order to achieve a reasonable soundness error. Consequently it is desirable to take $\mathcal{C}$ as large as possible. However, at the end of the protocol, V is only convinced that h^y is well formed, where $y = \mathrm{lcm}(1, \ldots, |\mathcal{C}|)$. So if V wants to perform operations on h which are returned to P, without risking leaking any information to P, V must raise h to the power y before proceeding. When plugged into the [CCL$^+$19] two-party EC-DSA protocol this entails raising a ciphertext to the power y at the end of the key generation phase. So $|\mathcal{C}|$ must be chosen small enough for this exponentiation to take reasonable time. Hence we set $\mathcal{C} := \{0,1\}^{10}$, and $y = \mathrm{lcm}(1, \ldots, 2^{10})$, which is a 1479 bits integer, so exponentiating to the power y remains efficient. To achieve a soundness error of $2^{-\lambda}$ the protocol must be repeated $\lambda/10$ times.

5.2 Assuming a standardised group

If we assume a standardised set up process, which allowed to provide a description of $\widehat{G}$, of the subgroups F and G_q and of a *random* generator g_q of G_q, one could completely omit the interactive set up phase for the CL encryption scheme and have all parties use the output of this standardised process. This significantly improves the $\mathsf{IKeyGen}$ protocol, as mentionned in Sec. 6.

Furthermore, assuming such a set up, we can replace the most expensive ZKPoK in [CCL$^+$19] by an argument of knowledge using similar techniques to those of Sec. 3.1, and relying on the strong root and low order assumptions in $\widehat{G}$. We detail this improvement in the following paragraph.

Efficient ZKAoK for the [CCL$^+$19] two party protocol. Castagnos et al. recently put forth a generic two party EC-DSA protocol from hash proof systems [CCL$^+$19]. They rely on a ZKPoK for the following relation:

$$\mathsf{R}_{\mathsf{CL-DL}} := \{(\mathsf{pk}, (c_1, c_2), Q); (x, r) \;\mid\; c_1 = g_q^r \wedge c_2 = f^x \mathsf{pk}^r \wedge Q = xP\}.$$

The interactive proof they provide for this relation uses binary challenges, consequently in order to achieve a satisfying soundness error of $2^{-\lambda}$, the proof must be repeated λ times.

Using similar techniques to those proposed in Sec. 3.1, and relying on the strong root and low order assumptions in $\widehat{G}$, we describe in Fig. 7 a much more efficient ZKAoK, which can be plugged into their overall protocol so as to further improve its' overall computational and communication costs.

We emphasise that in order for security of this proof to hold, g_q must be a random generator of G^q. Since the set up described in [CCL$^+$19] outputs a deterministic g_q, in order to plug the following proof in their protocol, we need

Setup:

1. $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F) \leftarrow \mathsf{Gen}(1^\lambda, q)$.
2. Let $\tilde{A} := \tilde{s} \cdot 2^{40}$, sample $t \xleftarrow{\$} [\tilde{A}]$ and let $g_q := \hat{g}_q^t$.

Input : (r, x) and $(\mathsf{pk}, c_1, c_2, Q, P)$		Input : $(\mathsf{pk}, c_1, c_2, Q, P)$
$r_1 \xleftarrow{\$} [\tilde{A} \cdot C \cdot 2^{40}]$		
$r_2 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$		
$t_1 := g_q^{r_1}$		
$t_2 := \mathsf{pk}^{r_1} f^{r_2}$		
$T := r_2 P$	$\xrightarrow{t_1, t_2, T}$	$k \xleftarrow{\$} \mathcal{C}$
	$\xleftarrow{k}$	
$u_1 := r_1 + k \cdot r \in \mathbf{Z}$		
$u_2 := r_2 + k \cdot x \in \mathbf{Z}/q\mathbf{Z}$	$\xrightarrow{u_1, u_2}$	$\mathtt{Check}\ u_1 \in [\tilde{A}C(2^{40}+1)]$; $u_2 \in \mathbf{Z}/q\mathbf{Z}$ and $g_q^{u_1} = t_1 \cdot (c_1)^k$ and $T + k \cdot Q = u_2 \cdot P$ and $\mathsf{pk}^{u_1} f^{u_2} = t_2 \cdot (c_2)^k$

Fig. 7: Zero-knowledge argument of knowledge for $\mathsf{R}_{\mathsf{CL-DL}}$.

to assume some standardised set up process as mentioned in 5.2. Theorem 5 states the security of the ZKAoK for $\mathsf{R}_{\mathsf{CL-DL}}$.

Theorem 5. *Let $\mathcal{C}$ be the challenge set for the interactive protocol of Fig. 7, and $C := |\mathcal{C}|$. If the strong root assumption is $(t'(\lambda), \epsilon_{SR}(\lambda))$-secure for Gen, and the C-low order assumption is $(t'(\lambda), \epsilon_{\mathsf{LO}}(\lambda))$-secure for Gen, denoting $\epsilon := \max(\epsilon_{SR}(\lambda), \epsilon_{\mathsf{LO}}(\lambda))$, then the interactive protocol of Fig. 7 is a computationally convincing proof of knowledge for $\mathsf{R}_{\mathsf{CL-DL}}$ with knowledge error κ, time bound t and failure probability $\nu(\lambda)$, where $\nu(\lambda) = 8\epsilon$, $t(\lambda) < t'(\lambda)/448$ and $\kappa(\lambda) = \max(4/C, 448t(\lambda)/t'(\lambda))$. If $r, x \in [\tilde{s} \cdot 2^{40}]$ (it is so when the prover is honest), the protocol is honest verifier statistical zero-knowledge.*

Proof. Completeness. If P knows $r \in [\tilde{A}]$ and $x \in \mathbf{Z}/q\mathbf{Z}$ s.t. $(\mathsf{pk}, (c_1, c_2), Q); (x, r) \in \mathsf{R}_{\mathsf{CL-DL}}$, and if both parties follow the protocol, one has $u_1 \in [\tilde{A}C(2^{40}+1)]$ and $u_2 \in \mathbf{Z}/q\mathbf{Z}$; $\mathsf{pk}^{u_1} f^{u_2} = \mathsf{pk}^{r_1 + k \cdot r} f^{r_2 + k \cdot x} = \mathsf{pk}^{r_1} f^{r_2} (\mathsf{pk}^r f^x)^k = t_2 \cdot (c_2)^k$; $u_2 \cdot P = (r_2 + k \cdot x)P = T + k \cdot Q$; and $g_q^{u_1} = g_q^{r_1 + k \cdot r} = t_1 \cdot (c_1)^k$.
Honest verifier zero-knowledge. Given pk, $\mathbf{c} = (c_1, c_2)$ and Q a simulator can sample $k \xleftarrow{\$} [C[$, $u_1 \xleftarrow{\$} [\tilde{A}C(2^{40}+1)]$ and $u_2 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$, compute $t_1 := g_q^{u_1} \cdot (c_1)^{-k}$, $t_2 := \mathsf{pk}^{u_1} \cdot f^{u_2} \cdot (c_2)^{-k}$ and $T := u_2 \cdot P - k \cdot Q$ such that the transcript $(t_1, t_2, T, k, u_1, u_2)$ is indistinguishable from a transcript produced by a real execution of the protocol where V runs on input $(\mathsf{pk}, c_1, c_2, Q, P)$.

[8] For correctness Bob also needs to multiply the signed message m' by $y \mod q$, during the signature algorithm.

Computational soundness. Let us analyse for which knowledge error functions $\kappa()$ the protocol of Fig. 7 satisfies the notion of soundness defined in Def. 3. Accordingly, let $\kappa()$ be any knowledge error function, such that $\kappa(\lambda) \geq 4/C$ for all λ. As in proof of thm. 2, consider a malicious prover P^*. Since there are C different challenges, if $\epsilon_{\mathsf{view},P} > \kappa(\lambda) \geq 4/C$, one can obtain in expected PT a situation where, for given (t_1, t_2, T), P^* has correctly answered two different challenges k and k'. Let Rewind be a (probabilistic) procedure that creates $k, k', u_1, u_2, u_1', u_2'$ in this way. We call u_1, u_2 and u_1', u_2' the corresponding answers, so we get:

$$g_q^{u_1-u_1'} = (c_1)^{k-k'}; \quad \mathsf{pk}^{u_1-u_1'} f^{u_2-u_2'} = (c_2)^{k-k'}; \text{ and } (u_2-u_2')P = (k-k')Q.$$

Since $k \neq k'$ and q is prime, with overwhelming probability $(1-1/q)$ it holds that $(k-k') \in \mathbf{Z}/q\mathbf{Z}^*$. In the following we assume this is the case[9].

Assume without loss of generality that $k > k'$ and suppose that $(k-k')$ divides (u_1-u_1') in $\mathbf{Z}$. We denote:

$$\nu_1 := g_q^{(u_1-u_1')/(k-k')} \cdot (c_1)^{-1} \text{ and } \nu_2 := \mathsf{pk}^{(u_1-u_1')/(k-k')} \cdot f^{(u_2-u_2')/(k-k')} \cdot (c_2)^{-1}.$$

Suppose that $\nu_1 = \nu_2 = 1$. Moreover, $V's$ check on the size of u_1, u_1' implies that $(u_1-u_1')/(k-k')$ is in the required interval. One can now easily verify that P^* knows $((\mathsf{pk}, \mathbf{c}, Q); ((u_2-u_2')/(k-k') \mod q, (u_1-u_1')/(k-k'))) \in \mathsf{R}_{\mathsf{CL-DL}}$, and from $k, k', u_1, u_2, u_1', u_2'$ one can thus extract a witness for the statement.

A set of values $k, k', u_1, u_2, u_1', u_2'$ is said to be bad if $k-k'$ divides u_1-u_1' but $\nu_1 \neq 1$ or $\nu_2 \neq 1$ or if $k-k'$ does not divide u_1-u_1'. The extractor M simply repeats calling Rewind (for this same $(\mathsf{pk}, \mathbf{c}, Q)$) until it gets a set of good values. The analysis of the knowledge soundness with this M and the polynomial $p(\lambda)$ from the definition set to the constant of 112 proceeds exactly as in proof of Thm. 2, such that there exists an algorithm $\mathcal{A}(P^*)$ that solves either the strong root problem for class groups with input $(\widehat{G}, \widehat{G}^q, g_q)$, or the low order problem in $\widehat{G}$ with probability $\mathsf{Adv}_{\kappa,M,p}(P^*, \lambda)/8$, and runs in time $448 \cdot t_{P^*}(k)/\kappa(\lambda)$ where $t_{P^*}(k)$ denotes the running time of P^*. The proof can finally be concluded as in proof of Thm. 2.

Remark 3. In [CCL+19], when proving the security of the overall two party EC-DSA protocol, the simulator must simulate the above proof of knowledge without knowing (r, x), to a malicious adversary. Consequently to be used in their protocol, the above ZKAoK must be secure against malicious adversaries.

In order to attain security against malicious verifiers V^*, which may deviate from the protocol, a simulator simulating P chooses a random $k_P \in \mathcal{C}$, computes t_1, t_2 and T as in the proof of zero-knowledge against honest verifiers, and sends them to V^*, *hoping* that the challenge k_V chosen by V^* will be s.t. $k_V = k_P$. If so the simulated view of V^* is indistinguishable from V^*'s view in a real execution, if not S rewinds V^* and starts again until $k_V = k_P$. Consequently for the simulation to run in polynomial time we cannot chose $\mathcal{C}$ arbitrarily big. In practice one could take $\mathcal{C} := [2^{40}]$ and repeat the protocol $\lambda/40$ times to achieve

[9] In fact in our application $C < q$, so this holds with probability 1.

a satisfying soundness error of $2^{-\lambda}$. We emphasise that this is still considerably better than the proof used in [CCL+19] for which $\mathcal{C} = \{0, 1\}$.

6 Efficiency comparisons

In this section, we analyse the theoretical complexity of our protocol by counting the number of exponentiations and communication of group elements. We compare the communication cost of our protocol to that of [GG18,LN18] for the standard NIST curves P-256, P-384 and P-521, corresponding to levels of security 128, 192 and 256. For the encryption scheme, we start with a 112 bit security, as in [GG18,LN18]' implementations, but also study the case where its level of security matches the security of the elliptic curves.

We chose to compare our work to best performing protocols using similar construction techniques (from homomorphic encryption) which achieve the same functionality, i.e. (t, n)-threshold ECDSA for any t s.t. $n \geq t+1$. We do not provide a comparison to [DKLs18,DKLs19] as they use OT which leads to protocols with a much higher communication cost. Similarly, and as noted in [DKO+19] a direct comparison to [DKO+19,SA19] is difficult as they rely on preprocessing to achieve efficient signing, which is a level of optimisation we have not considered. We don't compare to [GGN16,BGG17] as [GG18] is already faster and cheaper in terms of communication complexity.

The computed comm. cost is for our protocol as described in Sec. 3, and as such is provably secure. Conversely the implementation which [GG18] provided omits a number of range proofs present in their described protocol. Though this substantially improves the efficiency of their scheme, they themselves note that removing these proofs creates an attack which leaks information on the secret signing key shared among the servers. They conjecture this information is limited enough for the protocol to remain secure, however since no formal analysis is performed, the resulting scheme is not proven secure. For a fair comparison we estimate the comm. cost and timings of both their secure protocol and the stripped down version. In terms of bandwidth we outperform even their stripped down protocol.

In both protocols, when possible zero knowledge proofs are performed non interactively, replacing the challenge by a hash value, whose size depends on the security parameter λ. We note that our interactive set up for the CL encryption scheme uses a ZKPoK where challenges are of size 10bits (using the lcm trick), it must thus be repeated $\lambda/10$ times. We note however that the PoK of integer factorization used in the key generation of [GG18] has similar issues.

For non-malleable equivocable commitments, we use a cryptographic hash function H and define the commitment to x as $h = H(x, r)$, for a uniformly chosen r of length λ and assume that H behaves as a random oracle.

The comm. cost comparison is done by counting the number of bits that are both sent and received by a given party throughout the protocol[10]. In terms

[10] Broadcasting one element is counted as sending one element.

of timings, we count the number of exponentiations in the class group (for our protocol), the bit size of the exponent, and multiply this by 3/2 of the cost of a multiplication in the group. We compare this to an equivalent computation for [GG18], where we count exponentiations modulo N and N^2, the bit size of the exponent, and multiply this by 3/2 of the cost of a multiplication modulo N (resp. N^2). We do not count exponentiations and multiplications over the group of points of the elliptic curve as these are very cheap compared to the aforementioned computations, furthermore both protocols essentially perform identical operations on the curve.

The [LN18] protocol with Paillier encryption. We use the figures Lindell et al. provide in [LN18, Tab. 1] to compare our protocol to theirs. We note that – to their advantage – their key generation should include additional costs which are not counted in our figures (e.g. local Paillier key generation, verification of the ZKP of correctness of the Paillier key). The resulting costs are given in Tab.8a

The [GG18] protocol with Paillier encryption. The main cost in their key generation protocol is the ZKPoK of integer factorization, which is instantiated using [PS00, Thm. 8]. Precisely each prover commits to K values mod N, the challenge lives mod B, the final opening is an element of size A, where, as prescribed by Poupard and Stern, we take $\log(A) = \log(N)$, $\log(B) = \lambda$ and $K = \frac{\lambda + \log(|N|)}{log(C)}$ where $C := 2^{60}$ is chosen s.t. Floyd's cycle-finding algorithm is efficient in a space of size smaller than C. For their signature protocol, the cost of the ZK Proofs used in the MtA protocol are counted using [GG18, Appendix A].

The results are summarized in Fig. 8b. Since the range proofs (omitted in the stripped down version) only occur in the signing protocol, the timings and comm. cost of their interactive key generation is identical in both settings, we thus only provide these figures once. The comm. cost of each protocol is given in Bytes. The columns correspond to the elliptic curve used for EC-DSA, the security parameter λ in bits for the encryption scheme, the corresponding bit size of the modulus N, the timings of one Paillier exponentiation, of the key generation and of the signing phase and the total comm. in bytes for each interactive protocol. Modulus sizes are set according to the NIST recommendations.

Our protocol with the CL *encryption scheme.* For key generation we take into account the interactive key generation for the CL encryption scheme, which is done in parallel with $\mathsf{IKeyGen}$ s.t. the number of rounds of $\mathsf{IKeyGen}$ increases by only one broadcast per player. In $\mathsf{IKeyGen}$, each party performs 2 class group exponentiations of $\log(\tilde{s}) + 40$ bits (where $\tilde{s} \approx \sqrt{q \cdot \tilde{q}}$), to compute generators g_i and public keys pk_i, and $\lambda/10 \times n$ exponentiations of $\log(\tilde{s}) + 90$ bits for the proofs and checks in the ISetup sub-protocol.

Note that exponentiations in $\langle f \rangle$ are almost free. Signing uses $2 + 10t$ exponentiations of $\log(\tilde{s}) + 40$ bits (for computing ciphertexts and homomorphic operations), $2(t+1)$ of $\log(\tilde{s}) + 80 + \lambda$ (for the ZKAoK) and $2t$ exponentiations of size q (for homomorphic scalar multiplication of ciphertexts).

The results for our protocols are summarized in Fig. 8c. The columns correspond to the elliptic curve used for EC-DSA, the security parameter λ in bits for the encryption scheme, the corresponding fundamental discriminant $\Delta_K = -q \cdot \tilde{q}$ bit size, the timings of one class group exponentiation (for an exponent of $\lambda + 40$ bits, i.e. that used for encryption), of the key generation and of the signing phase and the total comm. in bytes for IKeyGen and ISign. The discriminant sizes are chosen according to [BJS10].

Rounds. In terms of the number of rounds, we perform identically to [LN18]. Our IKeyGen requires 5 rounds (only 4 assuming a standardised set up), compared to 4 in [GG18]. Our signing protocol requires 8 rounds as opposed to 9 in [GG18].

Curve	λ (bits)	N (bits)	Mult. (ms)	IKeyGen (ms)	ISign (ms)	IKeyGen (Bytes)	ISign (Bytes)
P-256	112	2048	0.0023	$> 52n + 52$	$99t$	$> 6\,336(n-1)$	$16\,064t$
P-256	128	3072	0.0048	$> 162n + 162$	$310t$	$> 9\,152(n-1)$	$22\,208t$
P-384	192	7680	0.0186	$> 1\,571n + 1571$	$3\,000t$	$> 22\,176(n-1)$	$51\,744t$
P-521	256	15360	0.0519	$> 8\,769n + 8\,769$	$16\,741t$	$> 43\,672(n-1)$	$99\,845t$

(a) [LN18]'s secure t out of n protocol.

				Provably secure (with range proofs)				Stripped down	
Curve	λ (bits)	N (bits)	Mult. (ms)	IKeyGen (ms)	ISign (ms)	IKeyGen (Bytes)	ISign (Bytes)	ISign (ms)	ISign (Bytes)
P-256	112	2048	0.0023	$64n + 7$	$140t$	$32(n+t) + 9\,990n - 64$	$23\,308t + 588$	$28t$	$4\,932t + 588$
P-256	128	3072	0.0048	$293n + 22$	$428t$	$32(n+t) + 21\,392n - 64$	$33\,568t + 608$	$88t$	$7\,008t + 608$
P-384	192	7680	0.0186	$7\,017n + 214$	$4\,071t$	$48(n+t) + 128088n - 96$	$81\,072t + 912$	$857t$	$16\,656t + 912$
P-521	256	15360	0.0519	$77\,725n + 1196$	$22\,528t$	$65(n+t) + 503\,591n - 130$	$159\,391t + 1\,232$	$4783t$	$32\,470t + 1\,231$

(b) [GG18]'s t out of n protocol.

Curve	λ (bits)	Δ_K (bits)	Mult. (ms)	IKeyGen (ms)	ISign (ms)	IKeyGen (Bytes)	ISign (Bytes)
P-256	112	1348	0.029	$366n + 62$	$430t + 137$	$\mathbf{32(n+t) + 2951n - 64}$	$\mathbf{3\,670t + 1\,747}$
P-256	128	1827	0.038	$744n + 109$	$730t + 237$	$\mathbf{32(n+t) + 4\,297n - 64}$	$\mathbf{4\,455t + 2\,052}$
P-384	192	3598	0.077	$4\,145n + 424$	$\mathbf{2\,780t + 903}$	$\mathbf{48(n+t) + 10\,851n - 96}$	$\mathbf{8\,022t + 3\,560}$
P-521	256	5971	0.137	$16\,432n + 1\,243$	$\mathbf{8\,011t + 2,608}$	$\mathbf{65(n+t) + 22\,942n - 130}$	$\mathbf{12\,576t + 5\,433}$

(c) Our secure t out of n protocol – With an interactive CL setup.

Fig. 8: Comparative sizes (in bits), timings (in ms) & comm. cost (in Bytes)

Comparison. Fig. 8 shows that the protocols of [LN18,GG18] are faster for both key generation and signing for standard security levels for the encryption scheme (112 and 128 bits of security) while our solution remains of the same order of magnitude. However for high security levels, our signing protocol is fastest from a 192-bits security level.

In terms of communications, our solution outperforms the other two protocols for all levels of security, factors vary according to the number of users n and the

desired threshold t. In terms of rounds, both our protocols use the same number of rounds as Lindell's. For key generation we use one more than [GG18], whereas for signing we use one less.

This situation can be explained by the following facts. Firstly with class groups of quadratic fields we can use lower parameters than with $\mathbf{Z}/n\mathbf{Z}$ (the best algorithm against the discrete logarithm problem in class groups has complexity $\mathcal{O}(L[1/2, o(1)])$ compared to an $\mathcal{O}(L[1/3, o(1)])$ for factoring). However, the group law is more complex in class groups, indeed exponentiations in class groups are cheaper than those modulo N^2 from the 192 bits level. So even if removing range proofs allows us to drastically reduce the number of exponentiations, our solution only takes less time from that level (while being of the same order of magnitude below this level).

We note that assuming a standardized set up for CL (as mentioned in Sec. 5.2), one would reduce the bandwidth consumption of $\mathsf{IKeyGen}$ by a factor varying from 6 to 16 (for increasing levels of security). Moreover in terms of timings, the only exponentiation in the class group would be each party computing its own ciphertext, and so the only operations linear in the number of users n would be on the curve (or integers modulo q), which are extremely efficient.

Acknowledgements. The authors would like to thank Rosario Gennaro and Steven Goldfeder for fruitful discussions. We also thank Omer Shlomovits for interesting insight on issues related to the practical implementation of threshold EC-DSA. This work was supported by the French ANR ALAMBIC project (ANR-16-CE39-0006). The research of Dario Catalano has been partially supported by the Università degli Studi di Catania, "Piano della Ricerca 2016/2018 — Linea di intervento 2"

References

BBBF18. D. Boneh, J. Bonneau, B. Bünz, and B. Fisch. Verifiable delay functions. In *CRYPTO 2018, Part I*, *LNCS* 10991, pages 757–788. Springer, Heidelberg, August 2018.

BBF18. D. Boneh, B. Bünz, and B. Fisch. A survey of two verifiable delay functions. Cryptology ePrint Archive, Report 2018/712, 2018. https://eprint.iacr.org/2018/712.

BBHM02. I. Biehl, J. Buchmann, S. Hamdy, and A. Meyer. A signature scheme based on the intractability of computing roots. *Designs, Codes and Cryptography*, 25(3):223–236, Mar 2002.

Bel04. K. Belabas. On quadratic fields with large 3-rank. *Mathematics of Computation*, 73(248):2061–2074, 2004.

BGG17. D. Boneh, R. Gennaro, and S. Goldfeder. Using level-1 homomorphic encryption to improve threshold dsa signatures for bitcoin wallet security. In *LATINCRYPT*, 2017.

BH01. J. Buchmann and S. Hamdy. A survey on IQ cryptography. In *Public Key Cryptography and Computational Number Theory*, pages 1–15. De Gruyter Proceedings in Mathematics, 2001.

BJS10. J.-F. Biasse, M. J. Jacobson, and A. K. Silvester. Security estimates for quadratic field based cryptosystems. In *ACISP 10*, *LNCS* 6168, pages 233–247. Springer, Heidelberg, July 2010.

Boy86. C. Boyd. Digital multisignature. *Cryptography and Coding*, pages 241–246, 1986.

Bue76. D. A. Buell. Class groups of quadratic fields. *Mathematics of Computation*, 30(135):610–623, 1976.

CCL+19. G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker. Two-party ECDSA from hash proof systems and efficient instantiations. In *CRYPTO 2019, Part III*, *LNCS* 11694, pages 191–221. Springer, Heidelberg, August 2019.

CH89. R. A. Croft and S. P. Harris. Public-key cryptography and reusable shared secret. *Cryptography and Coding*, pages 189–201, 1989.

CIL17. G. Castagnos, L. Imbert, and F. Laguillaumie. Encryption switching protocols revisited: Switching modulo p. In *CRYPTO 2017, Part I*, *LNCS* 10401, pages 255–287. Springer, Heidelberg, August 2017.

CKY09. J. Camenisch, A. Kiayias, and M. Yung. On the portability of generalized Schnorr proofs. In *EUROCRYPT 2009*, *LNCS* 5479, pages 425–442. Springer, Heidelberg, April 2009.

CL84. H. Cohen and H. W. Lenstra Jr. Heuristics on class groups. In *Number Theory*, pages 26–36, Berlin, Heidelberg, 1984. Springer Berlin Heidelberg.

CL15. G. Castagnos and F. Laguillaumie. Linearly homomorphic encryption from DDH. In *CT-RSA 2015*, *LNCS* 9048, pages 487–505. Springer, Heidelberg, April 2015.

CLT18. G. Castagnos, F. Laguillaumie, and I. Tucker. Practical fully secure unrestricted inner product functional encryption modulo p. In *ASIACRYPT 2018, Part II*, *LNCS* 11273, pages 733–764. Springer, Heidelberg, December 2018.

CS97. J. Camenisch and M. Stadler. Efficient group signature schemes for large groups (extended abstract). In *CRYPTO'97*, *LNCS* 1294, pages 410–424. Springer, Heidelberg, August 1997.

CS03. J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO 2003*, *LNCS* 2729, pages 126–144. Springer, Heidelberg, August 2003.

DDN00. D. Dolev, C. Dwork, and M. Naor. Nonmalleable cryptography. *SIAM Journal on Computing*, 30(2):391–437, 2000.

Des88. Y. Desmedt. Society and group oriented cryptography: A new concept. In *CRYPTO'87*, *LNCS* 293, pages 120–127. Springer, Heidelberg, August 1988.

DF90. Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *CRYPTO'89*, *LNCS* 435, pages 307–315. Springer, Heidelberg, August 1990.

DF02. I. Damgård and E. Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In *ASIACRYPT 2002*, *LNCS* 2501, pages 125–142. Springer, Heidelberg, December 2002.

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

DKLs19. J. Doerner, Y. Kondi, E. Lee, and a. shelat. Threshold ECDSA from ECDSA assumptions: The multiparty case. In *2019 IEEE Symposium on Security and Privacy*, pages 1051–1066. IEEE Computer Society Press, May 2019.

DKO+19. A. P. K. Dalskov, M. Keller, C. Orlandi, K. Shrishak, and H. Shulman. Securing dnssec keys via threshold ecdsa from generic mpc. *IACR Cryptology ePrint Archive*, 2019:889, 2019.

Fel87. P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proc. of FOCS 87*, pages 427–437. IEEE Computer Society, 1987.

GG18. R. Gennaro and S. Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In *ACM CCS 2018*, pages 1179–1194. ACM Press, October 2018.

GGN16. R. Gennaro, S. Goldfeder, and A. Narayanan. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In *ACNS 16*, *LNCS* 9696, pages 156–174. Springer, Heidelberg, June 2016.

Gil99. N. Gilboa. Two party RSA key generation. In *CRYPTO'99*, *LNCS* 1666, pages 116–129. Springer, Heidelberg, August 1999.

GJKR96a. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust and efficient sharing of RSA functions. In *CRYPTO'96*, *LNCS* 1109, pages 157–172. Springer, Heidelberg, August 1996.

GJKR96b. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. In *EUROCRYPT'96*, *LNCS* 1070, pages 354–371. Springer, Heidelberg, May 1996.

GMR88. S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

HS06. S. Hamdy and F. Saidak. Arithmetic properties of class numbers of imaginary quadratic fields. *JP Journal of Algebra, Number Theory and Application*, 6(1):129–148, 2006.

Lag80. J. Lagarias. Worst-case complexity bounds for algorithms in the theory of integral quadratic forms. *Journal of Algorithms*, 1(2):142 – 186, 1980.

Lin17. Y. Lindell. Fast secure two-party ECDSA signing. In *CRYPTO 2017, Part II*, *LNCS* 10402, pages 613–644. Springer, Heidelberg, August 2017.

Lip12. H. Lipmaa. Secure accumulators from euclidean rings without trusted setup. In *ACNS 12*, *LNCS* 7341, pages 224–240. Springer, Heidelberg, June 2012.

LN18. Y. Lindell and A. Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In *ACM CCS 2018*, pages 1837–1854. ACM Press, October 2018.

MR01. P. D. MacKenzie and M. K. Reiter. Two-party generation of DSA signatures. In *CRYPTO 2001*, *LNCS* 2139, pages 137–154. Springer, Heidelberg, August 2001.

MR04. P. D. MacKenzie and M. K. Reiter. Two-party generation of DSA signatures. *Int. J. Inf. Sec.*, 2(3-4):218–239, 2004.

Pie19. K. Pietrzak. Simple verifiable delay functions. In *ITCS 2019*, pages 60:1–60:15. LIPIcs, January 2019.

PR05. R. Pass and A. Rosen. Concurrent non-malleable commitments. In *46th FOCS*, pages 563–572. IEEE Computer Society Press, October 2005.

PS00. G. Poupard and J. Stern. Short proofs of knowledge for factoring. In *PKC 2000*, *LNCS* 1751, pages 147–166. Springer, Heidelberg, January 2000.

Que87. J. Quer. Corps quadratiques de 3-rang 6 et courbes elliptiques de rang 12. *C. R. Acad. Sci., Paris, Sér. I*, 305:215–218, 1987.

SA19. N. P. Smart and Y. T. Alaoui. Distributing any elliptic curve based protocol: With an application to mixnets. *IACR Cryptology ePrint Archive*, 2019:768, 2019.

Sch91. C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, January 1991.

SG98. V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In *EUROCRYPT'98*, *LNCS* 1403, pages 1–16. Springer, Heidelberg, May / June 1998.

Sha79. A. Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

Sho00. V. Shoup. Practical threshold signatures. In *EUROCRYPT 2000*, *LNCS* 1807, pages 207–220. Springer, Heidelberg, May 2000.

Van92. S. Vanstone. Responses to nist's proposal. *Communications of the ACM*, 35:50–52, July 1992. (communicated by John Anderson).

Wes19. B. Wesolowski. Efficient verifiable delay functions. In *EUROCRYPT 2019, Part III*, *LNCS* 11478, pages 379–407. Springer, Heidelberg, May 2019.