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, 33405 Talence, France
[2] Università di Catania, Catania, Italy
[3] Univ Lyon, EnsL, UCBL, CNRS, Inria, LIP, 69342 Lyon Cedex 07, France
ida.tucker@ens-lyon.fr
[4] Scuola Superiore di Catania, 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, MR01]) 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 Sect. 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[1].

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

---

[1] 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.

Classical tools that we use (Zero-knowledge proofs, Feldman verifiable secret sharing, Commitments) are described in the full version [CCL+20, Section 2.1].

2.1   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 1 (Existential unforgeability [GMR88]**).** *Consider a digital signature scheme* $\mathsf{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verif})$, *and a PPT algorithm* $\mathscr{A}$, *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* $\mathscr{A}$, *the probability* $\mathsf{Adv}^{\mathsf{eu\text{-}cma}}_{\mathsf{S}, \mathscr{A}}$ *that* $\mathscr{A}$ *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 $x_1, \ldots, x_n$ constitute a (t, n) threshold secret sharing of x.
$\mathsf{ISign}\langle (x_1, m); \ldots; (x_n, m) \rangle \to \langle (r, s) \rangle$ **or** $\langle \bot \rangle$ where $\bot$ 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 eu-cma: threshold unforgeability under chosen message attacks (tu-cma).

Definition 2 (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* $\mathsf{Adv}_{\mathsf{IS},\mathcal{A}}^{\mathsf{tu\text{-}cma}}$ *that $\mathcal{A}$ can produce a signature on a message $m \notin \mathcal{M}$ is a negligible function of λ.*

2.2   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 Subsect. 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 3 (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:*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{HSM}}(\lambda) := \left| 2 \cdot \Pr\left[b = b^\star : (\tilde{s}, f, \hat{g}_q, \widehat{G}, F) \leftarrow \mathsf{Gen}(1^\lambda, q), t \hookleftarrow \mathcal{D}_q, g_q = \hat{g}_q^t, \right. \right.$$

$$x \hookleftarrow \mathcal{D}, x' \hookleftarrow \mathcal{D}_q, b \xleftarrow{\$} \{0, 1\}, Z_0 \leftarrow g^x, Z_1 \leftarrow g_q^{x'},$$

$$\left. \left. b^\star \leftarrow \mathcal{A}(q, \tilde{s}, f, \hat{g}_q, g_q, \widehat{G}, F, Z_b, \mathsf{Solve}(.)) \right] - 1 \right|$$

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 $\hat{g}_q$.

Resulting 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 Sect. 3. We use the output of $\mathsf{Gen}(1^\lambda, q)$ and as in Definition 3, 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.*

2.3   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

Algo. KeyGen(pp)	**Algo.** Enc(pk, m)	**Algo.** Dec(sk, (c_1, c_2))
1. Pick sk $\hookleftarrow [\tilde{A}]$ and pk $:= g_q^{\text{sk}}$	1. Pick $r \hookleftarrow [\tilde{A}]$	1. Compute $M = c_2/c_1^{\text{sk}}$
2. Return (pk, sk)	2. Return $(g_q^r, f^m \text{pk}^r)$	2. Return Solve(M)

Fig. 1. Description of the CL encryption scheme

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 Sect. 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 4 (Low order assumption). *Consider a security parameter $\lambda \in$* **N***, and $\gamma \in$* **N***. The γ-low order problem (LOP$_\gamma$) is $(t(\lambda), \epsilon_{\text{LO}}(\lambda))$-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 $\epsilon_{\text{LO}}(\lambda)$. More precisely,*

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

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 adapted to 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 5 (Strong root assumption for Class Groups). *Consider a security parameter $\lambda \in$* **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 $(t(\lambda), \epsilon_{\text{SR}}(\lambda))$-secure for Gen if any adversary $\mathcal{A}$, running in time $\leq t(\lambda)$, solves the SRP with probability at most $\epsilon_{\text{SR}}(\lambda)$.*

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 these 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 $\frac{\eta_\infty(p)}{p^{r^2}\eta_r(p)^2}$ where $\eta_r(p) = \prod_{k=1}^r(1 - p^{-k})$. 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 tu-cma game (*cf.* Definition 2) – relies on the assumptions and tools introduced in Sect. 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 Sect. 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 Sect. 2.3). This ZKAoK is essential to secure our protocol against malicious adversaries. Next, in Sect. 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 IKeyGen protocol of threshold EC-DSA, thereby only increasing by one the number of rounds of the threshold signing protocol. Finally, in Sect. 3.3 we present our (t, n)-threshold EC-DSA signing protocol, whose security will be demonstrated in Sect. 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 $g_q := \hat{g}_q^t$.
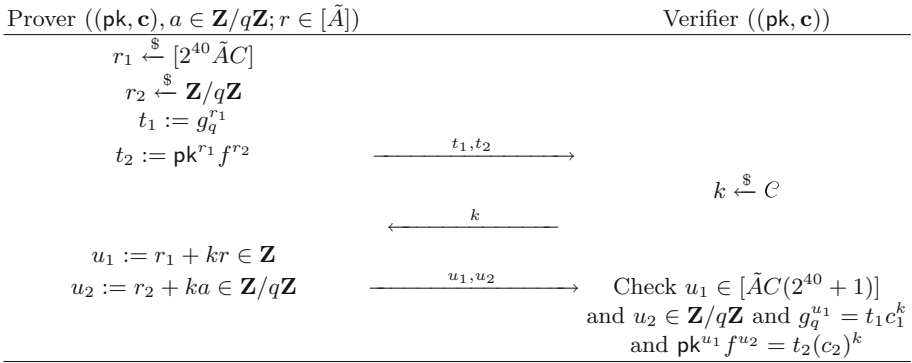
Prover $((\mathsf{pk}, \mathbf{c}), a \in \mathbf{Z}/q\mathbf{Z}; r \in [\tilde{A}])$		Verifier $((\mathsf{pk}, \mathbf{c}))$
$r_1 \xleftarrow{\$} [2^{40}\tilde{A}C]$		
$r_2 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$		
$t_1 := g_q^{r_1}$		
$t_2 := \mathsf{pk}^{r_1} f^{r_2}$	$\xrightarrow{\quad t_1, t_2 \quad}$	
		$k \xleftarrow{\$} \mathcal{C}$
	$\xleftarrow{\quad k \quad}$	
$u_1 := r_1 + kr \in \mathbf{Z}$		
$u_2 := r_2 + ka \in \mathbf{Z}/q\mathbf{Z}$	$\xrightarrow{\quad u_1, u_2 \quad}$	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 $(t'(\lambda), \epsilon_{SR}(\lambda))$-secure for Gen, and the C-low order assumption is $(t'(\lambda), \epsilon_{LO}(\lambda))$-secure for Gen, denoting $\epsilon := \max(\epsilon_{SR}(\lambda), \epsilon_{LO}(\lambda))$, then the interactive protocol of Fig. 2 is a computationally convincing proof of knowledge for $\mathsf{R}_{\mathsf{Enc}}$ 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 $r \in [\tilde{s} \cdot 2^{40}]$ (it is so when the prover is honest), the protocol is honest verifier statistical zero-knowledge.*

Proof. Computational soundness is proven in the full version [CCL+20, Thm. 2].

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 $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F) \leftarrow$ 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 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 Sect. 2.2 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 $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F) \leftarrow$ Gen$(\tilde{q}, q)$ to refer to this deterministic set up. We first define the functionality computed by ISetup, running in two steps.

Definition 6. *For a number of parties n,* ISetup *consists of the following interactive protocols:*

Step 1. $\langle k; \ldots; k \rangle \rightarrow \langle \tilde{q} \rangle$ **or** $\langle \bot \rangle$ *where $\bot$ is the error output, signifying the parties may abort the protocol, and $\tilde{q}$ is a random k bit prime.*
Step 2. $\langle (\tilde{q}, q); \ldots; (\tilde{q}, q) \rangle \rightarrow \langle (\tilde{s}, f, \hat{g}_q, \widehat{G}, F, g_q, t_1); \ldots; (\tilde{s}, f, \hat{g}_q, \widehat{G}, F, g_q, t_n) \rangle$ **or** $\langle \bot \rangle$ *where $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F) \leftarrow$ 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 perform 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 $(c_i, d_i) \leftarrow$ Com(r_i) and broadcasts c_i.
2. After receiving $\{c_j\}_{j \neq i}$, each P_i broadcasts d_i thus revealing r_i.
3. All players compute the common output $\tilde{q} := $ next-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 $g_i := \hat{g}_q^{t_i}$; $(\tilde{c}_i, \tilde{d}_i) \leftarrow$ Com(g_i), and broadcast $\tilde{c}_i$.
    (b) Receive $\{\tilde{c}_j\}_{j \neq i}$. Broadcast $\tilde{d}_i$ thus revealing g_i.
    (c) Perform a ZKPOK of t_i such that $g_i = \hat{g}_q^{t_i}$.[2] 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 $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F, g_q, t_i)$.

---

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

Theorem 3 states the security of the interactive protocol described in steps 1 and 2 above. The simulation and proof of indistinguishability are provided in the full version [CCL+20].

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 steps 1 and 2 described above securely compute* ISetup *with abort, in the presence of a malicious adversary corrupting any $t < n$ parties, with point-to-point channels.*

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 next-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. 3, and in phases 1, 3, 4, 5 of Fig. 4, 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. 4 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 IKeyGen, all parties run the ISetup protocol of Sect. 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 IKeyGen and ISetup are here described as two separate protocols, they can be ran in parallel. Consequently, in practice the number of rounds in 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 IKeyGen protocol (also depicted in Fig. 3) proceeds as follows:

1. Each P_i samples a random $u_i \overset{\$}{\leftarrow} \mathbf{Z}/q\mathbf{Z}$; 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 $(\mathsf{pk}_i, \mathsf{kgc}_i)$.
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
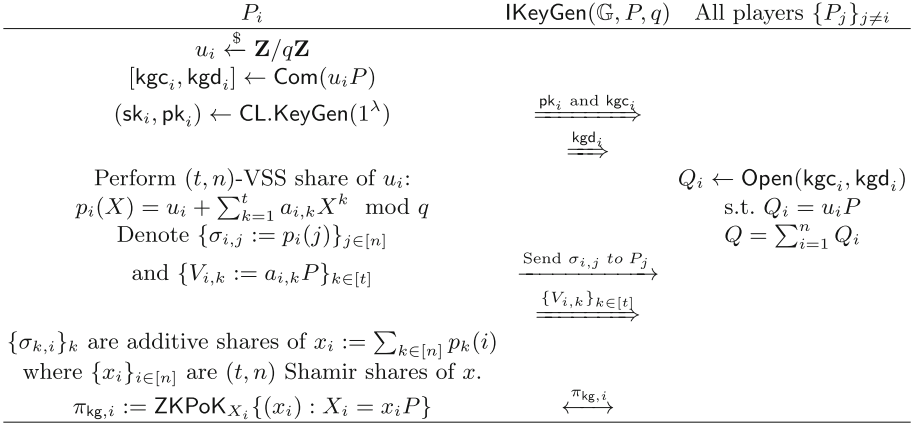
P_i	$\mathsf{IKeyGen}(\mathbb{G}, P, q)$	All players $\{P_j\}_{j\neq i}$
$u_i \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$		
$[\mathsf{kgc}_i, \mathsf{kgd}_i] \leftarrow \mathsf{Com}(u_i P)$		
$(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{CL.KeyGen}(1^\lambda)$	$\xrightarrow{\ \mathsf{pk}_i \text{ and } \mathsf{kgc}_i\ }$	
	$\xLongrightarrow{\ \mathsf{kgd}_i\ }$	
Perform (t,n)-VSS share of u_i:		$Q_i \leftarrow \mathsf{Open}(\mathsf{kgc}_i, \mathsf{kgd}_i)$
$p_i(X) = u_i + \sum_{k=1}^{t} a_{i,k} X^k \mod q$		s.t. $Q_i = u_i P$
Denote $\{\sigma_{i,j} := p_i(j)\}_{j\in[n]}$		$Q = \sum_{i=1}^{n} Q_i$
and $\{V_{i,k} := a_{i,k} P\}_{k\in[t]}$	$\xrightarrow{\ \text{Send } \sigma_{i,j} \text{ to } P_j\ }$	
	$\xLongrightarrow{\ \{V_{i,k}\}_{k\in[t]}\ }$	
$\{\sigma_{k,i}\}_k$ are additive shares of $x_i := \sum_{k\in[n]} p_k(i)$		
where $\{x_i\}_{i\in[n]}$ are (t,n) Shamir shares of x.		
$\pi_{\mathsf{kg},i} := \mathsf{ZKPoK}_{X_i}\{(x_i) : X_i = x_i P\}$	$\xleftrightarrow{\ \pi_{\mathsf{kg},i}\ }$	

Fig. 3. Threshold key generation

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. 3. 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. 4.

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:

(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.

P_i	Phase 1	All players $\{P_j\}_{j\neq i}$
$k_i, \gamma_i \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$		
$r_i \xleftarrow{\$} [\tilde{A}]$		
$c_{k_i} \leftarrow \mathsf{Enc}(\mathsf{pk}_i, k_i; r_i)$		
$[c_i, d_i] \leftarrow \mathsf{Com}(\gamma_i P)$	$\xrightarrow{c_i, c_{k_i}}$	if a proof fails, abort
$\pi_i := \mathsf{ZKAoK}_{\mathsf{pk}_i, c_{k_i}}\{(k_i, r_i) : ((\mathsf{pk}_i, c_{k_i}); (k_i, r_i)) \in \mathsf{R}_{\mathsf{Enc}}\}$	$\xleftarrow{\pi_i}$	
P_i	Phase 2	P_j
$\beta_{j,i}, \nu_{j,i} \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$		
$B_{j,i} := \nu_{j,i} \cdot P$		
$c_{\beta_{j,i}} \leftarrow \mathsf{Enc}(\mathsf{pk}_j, -\beta_{j,i})$		
$c_{\nu_{j,i}} \leftarrow \mathsf{Enc}(\mathsf{pk}_j, -\nu_{j,i})$		
$c_{k_j\gamma_i} \leftarrow \mathsf{EvalAdd}(\mathsf{EvalScal}(c_{k_j}, \gamma_i), c_{\beta_{j,i}})$		
$c_{k_j w_i} \leftarrow \mathsf{EvalAdd}(\mathsf{EvalScal}(c_{k_j}, w_i), c_{\nu_{j,i}})$	$\xrightarrow{c_{k_j\gamma_i}, c_{k_j w_i}, B_{j,i}}$	
		$\alpha_{j,i} \leftarrow \mathsf{Dec}(\mathsf{sk}_j, c_{k_j\gamma_i})$
		$\mu_{j,i} \leftarrow \mathsf{Dec}(\mathsf{sk}_j, c_{k_j w_i})$
		If $\mu_{j,i} \cdot P + B_{j,i} \neq W_i^{k_j}$ then abort
$\delta_i := k_i\gamma_i + \sum_{j\neq i}(\alpha_{i,j} + \beta_{j,i})$		
$\sigma_i := k_i w_i + \sum_{j\neq i}(\mu_{i,j} + \nu_{j,i})$		
P_i	Phase 3	All players $\{P_j\}_{j\neq i}$
	$\xrightarrow{\delta_i}$	$\delta = \sum_{i\in S} \delta_i = k\gamma$
P_i	Phase 4	All players $\{P_j\}_{j\neq i}$
	$\xrightarrow{d_i}$	$\Gamma_i := \mathsf{Open}(c_i, d_i)$
$\pi_{\gamma_i} := \mathsf{ZKPoK}_{\Gamma_i}\{(\gamma_i) : \Gamma_i = \gamma_i P\}$	$\xleftarrow{\pi_{\gamma_i}}$	if a proof fails, abort
		$R := \delta^{-1}(\sum_{i\in S} \Gamma_i)$ and $r := H'(R)$
P_i	Phase 5	All players $\{P_j\}_{j\neq i}$
$s_i := m k_i + r\sigma_i$		
$\ell_i, \rho_i \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$		
$V_i := s_i R + \ell_i P$ and $A_i := \rho_i P$		
$[\hat{c}_i, \hat{d}_i] \leftarrow \mathsf{Com}(V_i, A_i)$	$\xrightarrow{\hat{c}_i}$	
$\hat{\pi}_i := \mathsf{ZKPoK}_{(V_i, A_i)}\{(s_i, \ell_i, \rho_i) : V_i = s_i R + \ell_i P \wedge A_i = \rho_i P\}$	$\xrightarrow{\hat{d}_i}$	
	$\xleftarrow{\hat{\pi}_i}$	if a proof fails, abort
		$V := -mP - rQ + \sum_{i\in S} V_i$
		and $A := \sum_{i\in S} A_i$
$U_i := \rho_i V$ and $T_i := \ell_i A$		
$[\tilde{c}_i, \tilde{d}_i] \leftarrow \mathsf{Com}(U_i, T_i)$	$\xrightarrow{\tilde{c}_i}$	
	$\xrightarrow{\tilde{d}_i}$	if $\sum_{i\in S} T_i \neq \sum_{i\in S} U_i$ then abort.
	$\xrightarrow{s_i}$	$s := \sum_{i\in S} s_i$,
		if (r, s) is not a valid signature, abort,
		else return (r, s).

Fig. 4. Threshold signature protocol

(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 w_i as that used to compute 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 $[\hat{c}_i, \hat{d}_i] \leftarrow \mathsf{Com}(V_i, A_i)$. Each P_i broadcasts $\hat{c}_i$.

(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 $\mathscr{A}$ which breaks the threshold EC-DSA protocol of Figs. 3 and 4, then we can construct a forger $\mathscr{F}$ which uses $\mathscr{A}$ to break the unforgeability of standard EC-DSA. To this end $\mathscr{F}$ must simulate the environment of $\mathscr{A}$, so that $\mathscr{A}$'s view of its interactions with $\mathscr{F}$ are indistinguishable from $\mathscr{A}$'s view in a real execution of the protocol. Precisely, we show that if an adversary $\mathscr{A}$ corrupts $\{P_j\}_{j>1}$, one can construct a forger $\mathscr{F}$ simulating P_1 s.t. the output distribution of $\mathscr{F}$ is indistinguishable from $\mathscr{A}$'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). $\mathscr{F}$ 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, $\mathscr{F}$ 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 $\mathscr{F}$ must set up in its simulation with $\mathscr{A}$ this same public key Q (w/o knowing x). This will allow $\mathscr{F}$ to subsequently simulate interactively signing messages with $\mathscr{A}$, using the output of its' (standard) EC-DSA signing oracle.

The main differences with the proof of [GG18] arise from the fact $\mathscr{F}$ 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 $\mathscr{F}$ uses its' secret key does not compromise security, and we can still reduce the security of the protocol to the ind-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, $\mathscr{F}$ can not extract the decryption keys of corrupted players. The simulation is described below.

Simulating. P_1 in IKeyGen

1. $\mathscr{F}$ 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 ($\perp$ 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 $k_1, \gamma_1 \overset{\$}{\leftarrow} \mathbf{Z}/q\mathbf{Z}$ and $r_1 \overset{\$}{\leftarrow} [\tilde{A}]$ 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 $\mu_{j,1} \overset{\$}{\leftarrow} \mathbf{Z}/q\mathbf{Z}$ 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

$$\sum_{i > 1} \sigma_i = \sum_{i > 1} \left(k_i w_i + \sum_{j \neq i} \mu_{i,j} + \nu_{j,i} \right) = \sum_{i > 1} \sum_{j \neq i} (\mu_{i,j} + \nu_{j,i}) + \sum_{i > 1} k_i w_i$$

$$= \sum_{i > 1} (\mu_{i,1} + \nu_{1,i}) + \sum_{i > 1; j > 1} k_i w_j$$

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 $\mathscr{F}$ proceeds as follows:
  – $\mathscr{F}$ 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: $\mathscr{F}$ does the following
    • sample a random $\tilde{s}_1 \xleftarrow{\$} Zq$.
    • sample $\ell_1, \rho_1 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$, compute $V_1 := s_1 R + \ell_1 P$; $A_1 := \rho_1 P$; $[\widehat{c}_1, \widehat{d}_1] \leftarrow \mathsf{Com}(V_1, A_1)$ and send $\widehat{c}_1$ to $\mathscr{A}$.
    • receive $\{\widehat{c}_j\}_{j \neq 1}$ and decommit by broadcasting $\widehat{d}_1$. 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 $j \in S$, $j \neq 1$, $\mathscr{F}$ receive $\widehat{d}_j$ 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 $[\tilde{c}_1, \tilde{d}_1] \leftarrow \mathsf{Com}(U_1, T_1)$ and send $\tilde{c}_1$ to $\mathscr{A}$. Upon receiving $\{\tilde{c}_j\}_{j \neq 1}$ from $\mathscr{A}$, broadcast $\tilde{d}_1$ and receive the $\{\tilde{d}_j\}_{j \neq 1}$.
    • Now since $\sum_{i \in S} T_1 \neq \sum_{i \in S} U_1$ both $\mathscr{A}$ and $\mathscr{F}$ abort.

4.3   The Simulation of a Semi-correct Execution

Lemma 1. *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. $\mathscr{F}$ 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 $\widehat{G}$, $\mathscr{F}$ 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 $\mathscr{A}$ 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 fresh ciphertexts computed with pk_j follow identical distributions from $\mathscr{A}$'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 $\mathscr{A}$.
2. $\mathscr{F}$ 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 $\mathscr{F}$, 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 $\mathscr{F}$) 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 $\Gamma_1 = \gamma_1 P$, $\mathscr{F}$ is implicitly using $\widehat{k}_1 \neq k_1$, even though $\mathscr{A}$ received an encryption of k_1 in Phase 2. However, if $\mathscr{A}$ could tell apart a real and simulated execution based on this difference, one could use $\mathscr{A}$ to break the indistinguishabilty of the encryption scheme. So, under the assumption the CL encryption scheme is ind-cpa-secure, this change is unnoticeable to $\mathscr{A}$.

3. $\mathscr{F}$ 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 $\mathscr{F}$ 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 $\mathscr{F}$ 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 $\mathscr{A}$ 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 $\mathscr{F}$'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 $\mathscr{A}$ 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 2. *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 $\mathscr{F}$ (running P_1) and the adversary $\mathscr{A}$ (running all other players). In G_0, $\mathscr{F}$ runs the real protocol. The only change between G_0 and G_1 is that in G_1, $\mathscr{F}$ chooses U_1 as a random group element. In G_2 the simulator $\mathscr{F}$ runs the simulation described in Sect. 4.2.

Indistinguishability of G_0 and G_1. We prove that if there exists an adversary $\mathscr{A}_0$ distinguishing games G_0 and G_1, $\mathscr{A}_0$ 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 $\mathscr{F}_0$ runs $\mathscr{A}_0$, simulating the key generation phase s.t. $Q = \tilde{B}$. It does so by rewinding $\mathscr{A}_0$ in step 7 of the IKeyGen simulation and changing the decommitment of P_1 to $Q_1 := \tilde{B} - \sum_{j \in [n], j \neq 1} Q_j$. $\mathscr{F}_0$ also extracts the values $\{x_j\}_{j \in [n], j \neq 1}$ chosen by $\mathscr{A}_0$ from the ZKPoK of step 11 of the IKeyGen simulation. Note that at this point $Q = \tilde{B}$ and $\mathscr{F}_0$ knows x_i and the decryption key sk_1 matching pk_1, but not b and therefore not x_1.

Next $\mathscr{F}_0$ 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 $\mathscr{F}_0$ 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 $\mathscr{F}_0$) s.t. $w_1 = b - w_A$. $\mathscr{F}_0$ 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 $\mathscr{F}_0$ 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 $\mathscr{F}_0$ 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 $\mathscr{F}_0$ 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 $\mathscr{F}_0$. This allows $\mathscr{F}_0$ to compute the correct value $\sigma_1 \cdot P = \tilde{k} B + \mu_1 \cdot P$ 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 $\mathscr{F}_0$.

In the simulation of Phase 5, $\mathscr{F}_0$ 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 $\mathscr{A}_0$'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 $\mathscr{F}_0$ 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 $\mathscr{F}_0$ 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, $\mathscr{F}$ 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 $\mathscr{F}_0$ 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 $\mathscr{F}_0$ 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 Sect. 4.2 the forger $\mathscr{F}$ simulating $\mathscr{A}$'s environment can detect whether we are in a semi-correct-execution or not, i.e. whether $\mathscr{A}$ decides to be malicious and terminate the protocol with an invalid signature. Consequently $\mathscr{F}$ always knows how to simulate $\mathscr{A}$'s view and all simulations are indistinguishable of real executions of the protocol. Moreover if $\mathscr{A}$, having corrupted up to t parties in the threshold EC-DSA protocol, outputs a forgery, since $\mathscr{F}$ set up with $\mathscr{A}$ the same public key Q as it received from its' EC-DSA challenger, $\mathscr{F}$ can use this signature as its own forgery, thus breaking the existential unforgeability of standard EC-DSA.

Denoting $\mathsf{Adv}^{\mathsf{tu\text{-}cma}}_{\Pi, \mathscr{A}}$, $\mathscr{A}$'s advantage in breaking the existential unforgeability of our threshold protocol, and $\mathsf{Adv}^{\mathsf{eu\text{-}cma}}_{\mathsf{ecdsa}, \mathscr{F}}$ the forger $\mathscr{F}$'s advantage in breaking the existential unforgeability of standard EC-DSA, from Lemmas 1 and 2 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: $|\mathsf{Adv}^{\mathsf{eu\text{-}cma}}_{\mathsf{ecdsa}, \mathscr{F}} - \mathsf{Adv}^{\mathsf{tu\text{-}cma}}_{\Pi, \mathscr{A}}| \leq \mathsf{negl}(\lambda)$. Under the security of the EC-DSA signature scheme, $\mathsf{Adv}^{\mathsf{eu\text{-}cma}}_{\mathsf{ecdsa}, \mathscr{F}}$ must be negligible, which implies that $\mathsf{Adv}^{\mathsf{tu\text{-}cma}}_{\Pi, \mathscr{A}}$ should too, thus contradicting the assumption that $\mathscr{A}$ 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 existentially unforgeable; 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 Figs. 3 and 4 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 Sect. 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. 5 is a zero knowledge proof of knowledge for the following relation:

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

$$
\begin{array}{ccc}
\underline{P(x, h := g_q^x)} & \text{public: } g_q & \underline{V(h)} \\
r \xleftarrow{\$} [0, \tilde{s} \cdot 2^{90}] & & \\
t := g_q^r & \xrightarrow{\quad t \quad} & \\
& & k \xleftarrow{\$} \{0,1\}^{10} \\
& \xleftarrow{\quad k \quad} & \\
u := r + kx \in \mathbf{Z} & \xrightarrow{\quad u \quad} & \\
& & \text{Check } g_q^u = t \cdot h^k
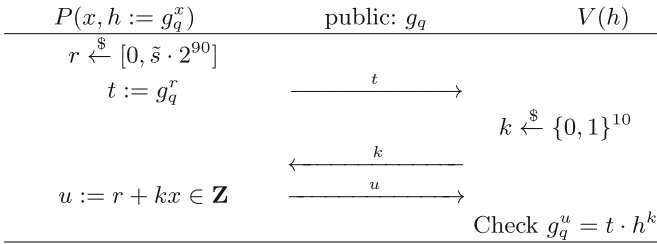\end{array}
$$

Fig. 5. ZKPoK of z s.t. $h^y = g_q^z$ where $y = \mathrm{lcm}(1, 2, 3, \ldots, 2^{10})$

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 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 Sect. 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 Sect. 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:

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

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:

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

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[3]. 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 fixes the number of protocol repetitions required to achieve a reasonable soundness error. It is thus 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

---

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

scheme and have all parties use the output of this standardised process. This significantly improves the IKeyGen protocol, as mentionned in Sect. 6.

Furthermore, assuming such a set up, we can replace the most expensive ZKPoK in [CCL+19] by an argument of knowledge for the same relation using similar techniques to those of Sect. 3.1, and relying on the strong root and low order assumptions in $\widehat{G}$. The resulting ZKAoK and a proof of its security are provided in the full version of this article [CCL+20, Section 5.2].

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][4] 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 the implementations of [GG18,LN18], but also study the case where its level of security matches the security of the elliptic curves.

The computed comm. cost is for our provably secure protocol as described in Sect. 3. 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.

---

[4] These are the best performing protocols using similar construction techniques to us (from homomorphic encryption), and achieving the same functionality, i.e. (t, n)-threshold ECDSA for any t s.t. $n \geq t + 1$. We do not compare 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 comm. cost comparison is done by counting the number of bits that are both sent and received by a given party throughout the protocol[5]. In terms 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, Table 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 Fig. 6a

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, Theorem 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. 6b. 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 CL *Encryption.* For key generation we take into account the interactive key generation for the CL encryption scheme, which is done in parallel with IKeyGen s.t. the number of rounds of IKeyGen increases by only one broadcast per player. In 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

---

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

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. 6c. 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. 6. Comparative sizes (in bits), timings (in ms) & comm. cost (in Bytes)

Comparison. Figure 6 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 our signing protocol is fastest from a 192-bits security level. In terms of communication, our solution outperforms both protocols for all security levels, factors vary according to the number of users n and the threshold t. In terms of rounds, our protocols use the same number of rounds as Lindell's. For key generation we use one more than [GG18], 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 Sect. 5.2), one would reduce the bandwidth consumption of 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.

References

[BBBF18] Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part I. LNCS, vol. 10991, pp. 757–788. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96884-1_25

[BBF18] Boneh, D., Bünz, B., Fisch, B.: A survey of two verifiable delay functions. Cryptology ePrint Archive, Report 2018/712 (2018)

[BBHM02] Biehl, I., Buchmann, J., Hamdy, S., Meyer, A.: A signature scheme based on the intractability of computing roots. Des. Codes Crypt. **25**(3), 223–236 (2002)

[Bel04] Belabas, K.: On quadratic fields with large 3-rank. Math. Comput. **73**(248), 2061–2074 (2004)

[BGG17] Boneh, D., Gennaro, R., Goldfeder, S.: Using level-1 homomorphic encryption to improve threshold DSA signatures for bitcoin wallet security. In: Lange, T., Dunkelman, O. (eds.) LATINCRYPT 2017. LNCS, vol. 11368, pp. 352–377. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25283-0_19

[BH01] Buchmann, J., Hamdy, S.: A survey on IQ cryptography. In: Public Key Cryptography and Computational Number Theory. De Gruyter Proceedings in Mathematics (2001)

[BJS10] Biasse, J.-F., Jacobson, M.J., Silvester, A.K.: Security estimates for quadratic field based cryptosystems. In: Steinfeld, R., Hawkes, P. (eds.) ACISP 2010. LNCS, vol. 6168, pp. 233–247. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14081-5_15

[Boy86] Boyd, C.: Digital multisignature. In: Cryptography and Coding (1986)

[Bue76] Buell, D.A.: Class groups of quadratic fields. Math. Comput. **30**(135), 610–623 (1976)

[CCL+19] Castagnos, G., Catalano, D., Laguillaumie, F., Savasta, F., Tucker, I.: Two-party ECDSA from hash proof systems and efficient instantiations. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 191–221. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_7

[CCL+20] Castagnos, G., Catalano, D., Laguillaumie, F., Savasta, F., Tucker, I.: Bandwidth-efficient threshold EC-DSA. Cryptology ePrint Archive, Report 2020/084 (2020)

[CH89] Croft, R.A., Harris, S.P.: Public-key cryptography and reusable shared secret. In: Cryptography and Coding (1989)

[CIL17] Castagnos, G., Imbert, L., Laguillaumie, F.: Encryption switching protocols revisited: switching modulo p. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 255–287. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_9

[CKY09] Camenisch, J., Kiayias, A., Yung, M.: On the portability of generalized Schnorr proofs. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 425–442. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01001-9_25

[CL84] Cohen, H., Lenstra, H.W.: Heuristics on class groups. In: Chudnovsky, D.V., Chudnovsky, G.V., Cohn, H., Nathanson, M.B. (eds.) Number Theory. LNM, vol. 1052, pp. 26–36. Springer, Heidelberg (1984). https://doi.org/10.1007/BFb0071539

[CL15] Castagnos, G., Laguillaumie, F.: Linearly homomorphic encryption from DDH. In: Nyberg, K. (ed.) CT-RSA 2015. LNCS, vol. 9048, pp. 487–505. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16715-2_26

[CLT18] Castagnos, G., Laguillaumie, F., Tucker, I.: Practical fully secure unrestricted inner product functional encryption modulo p. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part II. LNCS, vol. 11273, pp. 733–764. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03329-3_25

[CS97] Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups. In: Kaliski, B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 410–424. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0052252

[CS03] Camenisch, J., Shoup, V.: Practical verifiable encryption and decryption of discrete logarithms. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 126–144. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_8

[DDN00] Dolev, D., Dwork, C., Naor, M.: Nonmalleable cryptography. SIAM J. Comput. **30**(2), 391–437 (2000)

[Des88] Desmedt, Y.: Society and group oriented cryptography: a new concept. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 120–127. Springer, Heidelberg (1988). https://doi.org/10.1007/3-540-48184-2_8

[DF90] Desmedt, Y., Frankel, Y.: Threshold cryptosystems. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 307–315. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_28

[DF02] Damgård, I., Fujisaki, E.: A statistically-hiding integer commitment scheme based on groups with hidden order. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 125–142. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-36178-2_8

[DKLs18] Doerner, J., Kondi, Y., Lee, E., Shelat, A.: Secure two-party threshold ECDSA from ECDSA assumptions. In: 2018 IEEE Symposium on Security and Privacy. IEEE Computer Society Press (2018)

[DKLs19] Doerner, J., Kondi, Y., Lee, E., Shelat, A.: Threshold ECDSA from ECDSA assumptions: the multiparty case. In: 2019 IEEE Symposium on Security and Privacy. IEEE Computer Society Press (2019)

[DKO+19] Dalskov, A.P.K., Keller, M., Orlandi, C., Shrishak, K., Shulman, H.: Securing DNSSEC keys via threshold ECDSA from generic MPC. IACR Cryptology ePrint Archive, 2019:889 (2019)

[Fel87] Feldman, P.: A practical scheme for non-interactive verifiable secret sharing. In: Proceedings of FOCS 1987. IEEE Computer Society (1987)

[GG18] Gennaro, R., Goldfeder, S.: Fast multiparty threshold ECDSA with fast trustless setup. In: ACM CCS 2018. ACM Press (2018)

[GGN16] Gennaro, R., Goldfeder, S., Narayanan, A.: Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In: Manulis, M., Sadeghi, A.-R., Schneider, S. (eds.) ACNS 2016. LNCS, vol. 9696, pp. 156–174. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39555-5_9

[Gil99] Gilboa, N.: Two party RSA key generation. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 116–129. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_8

[GJKR96a] Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Robust and efficient sharing of RSA functions. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 157–172. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_13

[GJKR96b] Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Robust threshold DSS signatures. In: Maurer, U. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 354–371. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68339-9_31

[GMR88] Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Comput. **17**(2), 281–308 (1988)

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

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

[Lin17] Lindell, Y.: Fast secure two-party ECDSA signing. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part II. LNCS, vol. 10402, pp. 613–644. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63715-0_21

[Lip12] Lipmaa, H.: Secure accumulators from euclidean rings without trusted setup. In: Bao, F., Samarati, P., Zhou, J. (eds.) ACNS 2012. LNCS, vol. 7341, pp. 224–240. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31284-7_14

[LN18] Lindell, Y., Nof, A.: Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In: ACM CCS 2018. ACM Press (2018)

[MR01] MacKenzie, P.D., Reiter, M.K.: Two-party generation of DSA signatures. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 137–154. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_8

[Pie19] Pietrzak, K.: Simple verifiable delay functions. In: ITCS 2019. LIPIcs (2019)

[PR05] Pass, R., Rosen, A.: Concurrent non-malleable commitments. In: 46th FOCS. IEEE Computer Society Press (2005)

[PS00] Poupard, G., Stern, J.: Short proofs of knowledge for factoring. In: Imai, H., Zheng, Y. (eds.) PKC 2000. LNCS, vol. 1751, pp. 147–166. Springer, Heidelberg (2000). https://doi.org/10.1007/978-3-540-46588-1_11

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

[SA19] Smart, N.P., Alaoui, Y.T.: Distributing any elliptic curve based protocol: with an application to MixNets. IACR Cryptology ePrint Archive 2019:768 (2019)

[Sch91] Schnorr, C.-P.: Efficient signature generation by smart cards. J. Cryptol. **4**(3), 161–174 (1991)

[SG98] Shoup, V., Gennaro, R.: Securing threshold cryptosystems against chosen ciphertext attack. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 1–16. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054113

[Sha79] Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)

[Sho00] Shoup, V.: Practical threshold signatures. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 207–220. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45539-6_15

[Van92] Vanstone, S.: Responses to NIST's proposal. Commun. ACM **35**, 41–54 (1992). (communicated by John Anderson)

[Wes19] Wesolowski, B.: Efficient verifiable delay functions. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part III. LNCS, vol. 11478, pp. 379–407. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17659-4_13