Bandwidth-efficient threshold EC-DSA revisited: Online/Offline Extensions, Identifiable Aborts Proactive and Adaptive Security

Abstract. Due to their use in crypto-currencies, threshold ECDSA signatures have received much attention in recent years. Though efficient solutions now exist both for the two party, and the full threshold scenario, there is still much room for improvement, be it in terms of protocol functionality, strengthening security or further optimising efficiency.
In the past few months, a range of protocols have been published, allowing for a non interactive – and hence extremely efficient – signing protocol; providing new features, such as identifiable aborts (parties can be held accountable if they cause the protocol to fail), fairness in the honest majority setting (all parties receive output or nobody does) and other properties. In some cases, security is proven in the strong simulation based model. We combine ideas from the aforementioned articles with the suggestion of Castagnos *et al.* (PKC 2020) to use the class group based CL framework so as to drastically reduce bandwidth consumption. Building upon this latter protocol we present a new, maliciously secure, full threshold ECDSA protocol that achieving additional features without sacrificing efficiency. Our most basic protocol boasts a non interactive signature algorithm and identifiable aborts. We also propose a more advanced variant that also achieves adaptive security (for the n-out-of-n case) and proactive security. Our resulting constructions improve upon state of the art Paillier's based realizations achieving similar goals by up to a 10 factor in bandwidth consumption.

1 Introduction

Threshold signatures allow a set of n mutually mistrusting users to share a common signing key sk and to issue signatures as long as a quorum of them decides to do so. A key feature of this primitive is that at no stage during the signing process do users need to explicitly reconstruct the signing key. More in detail, the threshold $t < n$ establishes that $t + 1$ users can collaboratively sign, whereas any coalition of t or less participants can not; in fact, an adversary corrupting up to t users gets no information whatsoever about sk. In practice, this makes threshold signatures (and more generally threshold cryptography) a very useful tool to significantly reduce the losses caused by security break ins.

Threshold signatures have been extensively studied over the last thirty years (e.g. [Des88,GJKR96b,Sho00,Boy86,CH89,MR04]) but recently significant attention has been devoted to the specific case of threshold EC-DSA signatures. There are several reasons for this. First EC-DSA is the signature scheme used in Bitcoin and other cryptocurrencies. In many of these, knowing the secret signing key corresponding to an identity equates to owning the associated coins. Hence secure threshold variants of EC-DSA are an effective means of preventing Bitcoin thefts: rather than storing sk in a single location, one distributes it over several servers so that $t+1$ of them must be compromised in order leak any useful information.

Clearly, in order for secure solutions to be relevant in these settings, they need to be efficient, both in terms of overall computation and in terms of communication costs induced by the protocols, and indeed practical efficiency was the main missing ingredient of state of the art solutions (e.g. [GJKR96a,MR01]) when the cryptocurrency era began.

Over the last five years, many improved solutions have been presented both for the two party [Lin17,DKLs18,CCL$^+$19] and for the more general t-out-of-n case [GGN16,GG18,LN18,DKLs19,CCL$^+$20,CGG$^+$20]). Among the latter solutions, a very recent proposal by Canetti et al. [CGG$^+$20] departs from previous realizations in that it offers several additional features. It handles adaptive adversaries, achieves accountability (by identifying corrupted parties in case of problems when generating the signature) and uses an interesting online/offline optimization where one can perform the heaviest part of the computation before the message to be signed is known. Their resulting protocols are UC secure and allow for non interactive online signing (assuming the offline preprocessing). Still they are considerably heavier than previous solutions (e.g. [CCL$^+$20]) especially in terms of overall communication cost. This is particularly annoying as, in practice, bandwidth consumption tends to be one of the main concerns.

Our contribution. In this paper we present new, practical, threshold variants of the EC-DSA signature scheme that are provably secure (but not UC secure); realize both accountability and online/offline efficiency (in the sense discussed above). As detailed at the end of the article, our solutions reduce the bandwidth consumption of [CGG$^+$20] by up to a factor 10.

Our most basic construction works for any $t < n$ and is proved secure against static adversaries. Next, following Canetti et al. [CGG$^+$20] we show how to generalize it to the setting of adaptive adversaries for the special case when $t = n-1$.[1] The protocol can also easily be modified to achieve proactive security.

Details of our contribution. At a very high level, our constructions build upon the bandwidth efficient threshold EC-DSA scheme recently proposed by Castagnos *et al.* in [CCL$^+$20] which, in turn, revisits the threshold EC-DSA protocol by Gennaro and Goldfeder [GG18]. Recall that in EC-DSA signatures the public key is an elliptic curve point Q and the signing key is a scalar x, such that $Q = xP$, where 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 computes (for a random $k \in \mathbf{Z}/q\mathbf{Z}$) the point $R = k^{-1}P$. Next,

[1] Note that the whole work of Canetti *et al.* focuses on the n-out-of-n case and does not explicitly consider more general (t, n)-thresholds. They informally suggest directions to extend their work to the latter setting. We suspect that these methodologies apply to our protocols as well but we leave these extensions to future work.

letting $r = r_x \bmod q$ - where $R = (r_x, r_y)$ - one sets $s = kH(m) + krx \bmod q$. The signature is the pair (r, s).

The notoriously hard part, when trying to come up with a threshold variant, comes from the fact that one has to (distributively) compute both $R = k^{-1}P$ and a multiplication of the shared secrets k, x. Gennaro and Goldfeder [GG18] (and Castagnos *et al.* [CCL+20]) address this multiplication step as follows. Given two secrets $a = a_1 + \ldots + a_n$, $b = b_1 + \ldots + b_n$ additively shared among the players (i.e. player P_i holds a_i, b_i), players compute the product ab by computing additive shares of each monomial $a_i b_j$. In particular, this is done via a simple protocol by Gilboa[Gil99] that uses a linearly homomorphic encryption as underlying building block. Castagnos *et al.* [CCL+20] manage to make this step particularly efficient, in terms of bandwidth consumption, by relying on the Castagnos-Laguillaumie linearly homomorphic cryptosystem [CL15] (CL from now on). In a nutshell, the gain in efficiency arises from the fact that CL relies on class groups of imaginary quadratic fields and, in particular, this allows to use $\mathbf{Z}/q\mathbf{Z}$ as underlying message space, where q is the same large prime used in EC-DSA signatures. When used appropriately, this allows to avoid range proofs and other inefficiencies induced by the use of Paillier's cryptosystem in [GG18].

To allow online/offline efficiency in the protocol sketched above we basically adapt an idea originally proposed by Gennaro and Goldfeder [GG20][2] to the Castagnos *et al.* protocol. Informally, the players compute shares of k^{-1} and kx in the offline phase. Then, once the message m becomes available, players (non-interactively) compute the remaining part of the signature by broadcasting the share $s_i = k_i H(m) + r(kx)_i$. Notice that each player can locally perform this computation using publicly known information only. We remark that in order for this mechanism to be secure we need to assume that EC-DSA remains secure even when the adversary is allowed to see R before choosing the message m. While this is a seemingly stronger (but still reasonable) assumption than standard EC-DSA, it is clearly necessary in online/offline setting,

Introducing accountability on the other hand requires more work. Clearly an obvious way to identify players that behave maliciously would be to force them to prove in zero knowledge that they followed the protocol correctly. The problem with this solution is that it typically induces significant communication overhead, thus making the resulting protocol impractical. Gennaro and Goldfeder, in [GG20], follow a different approach specifically tailored to their online/offline solution. Informally, they argue that, in their protocol, malicious players can be identified easily both in the online and in the offline phase. In the online phase one identifies misbehaviors by simply checking against public information the shares of the component s that each player provides.

For the offline phase, their key observation is that, since no signature has been produced yet, players can simply reveal the random choices they used during the whole protocol, thus making their behavior verifiable. Notice that this means players also need to reveal the randomness used to create the ciphertexts

[2] This result currently appears as part of the [CGG+20] paper, but it was originally made public as a separate result.

they produced. When trying to translate this latter strategy to the setting of Castagnos *et al.*'s protocol [CCL+20] a technical problem prevents the resulting proof to go through. Informally this has to do with the fact that the security of their protocol crucially relies on some properties of hash proof systems.[3] The technical complication in the proof arises when the simulator needs to switch from valid ciphertexts to invalid ones that perfectly hide the underlying plaintext. This step is crucial for the proof in [CCL+20] to go through but becomes problematic here as the simulator cannot provide the randomness used to create the ciphertexts, as no valid ciphertexts exist anymore!

We circumvent this difficult by designing new zero-knowledge proofs that manage to let the simulator complete the proof without compromising the overall efficiency of the protocol.

As a final contribution, we propose a variant of our protocol that achieves adaptive security for the n-out-of-n case. This solution admits a simple extension which also encompasses proactive security. The most interesting feature of this latter protocol is that, for numbers of players one would expect in real life applications, it is very efficient, much more efficient than the corresponding protocol from [CGG+20]. This is due to the fact both solutions rely on a key refresh/setup protocol that, among other things, explicitly requires players to generate new encryption key pairs. In our case it is enough to generate a couple of the form $(h_i = g^{\mathsf{sk}})$ where g is a public, fixed parameter whereas in [CGG+20] each player is required to generate a new Paillier (RSA) modulus N together with a proof that it has been constructed correctly. Concretely, for $n = 5$ and a 112-bit level of security, the total data sent and received between players in our key refresh protocol is *15 times less* than theirs (28 KBytes as opposed to 420 KBytes).

Efficiency comparisons. We compare the communication cost of our signature protocol to those of Canetti et al. for the standard NIST curve P-256 corresponding to a 128 bit security level. Regarding encryption, 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 curve. In both cases, our comparison shows that our signing protocol is an order of magnitude more efficient.

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 $\mathsf{IP}\langle x_1; \ldots; x_n\rangle \to \langle y_1; \ldots; y_n\rangle$ 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.

[3] The connection with hash proof systems comes from the fact that the underlying linearly homomorphic encryption in [CCL+20] scheme is the one resulting from hash proof systems when using CL as underlying building block

2.1 Tools

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

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

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

Feldman verifiable secret sharing. A verifiable secret sharing (VSS) protocol allows to share a secret between n parties in a verifiable way. In our protocol we use Feldman's VSS [Fel87]; details for which can be found in Appx I.

Commitments. Our protocol uses non interactive equivocal commitments; a more formal description of which can be found in Appx II, we here only provide the syntax. These consist of algorithms $\mathsf{Setup}(1^\lambda) \to (\mathsf{pp}, \mathsf{tk})$ outputting public parameters pp and secret trapdoor key tk; $\mathsf{Com}(m, r) \to [\mathsf{c}(m), \mathsf{d}(m)]$ which on input a message m and random coins r, outputs the commitment $\mathsf{c}(m)$ and an opening value $\mathsf{d}(m)$ (refusal to open leads to $\mathsf{d}(m) = \bot$); (3) $\mathsf{Open}(\mathsf{c}, \mathsf{d}) \to m$ or $\bot$ which on input a commitment c and an opening value d, outputs either a message m or an error symbol $\bot$; and (4) $\mathsf{Equiv}(\mathsf{tk}, m, r, m') \to \widehat{\mathsf{d}}$.

2.2 The elliptic curve digital signature algorithm

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

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

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

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

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

As in [CGG$^+$20] we assume a somewhat enhanced variant of existential unforgeability under chosen message attacks (e-eu-cma) for EC-DSA. In this notion, for each signature query performed by the adversary $\mathcal{A}$, it gets to see the randomness R used to sign before choosing the message to be signed.

Definition 1 (Enhanced existential unforgeability [CGG$^+$20]). *Consider a PPT algorithm $\mathcal{A}$, which is given as input a verification key Q output by* $\mathsf{KeyGen}(\mathbb{G}, q, P) \to (x, Q)$ *and access to oracles:*

– $\mathcal{O}^R$ *to obtain a uniformly random point $R = (r_x, r_y)$ in $\mathbb{G}$;*
– $\mathcal{O}^{\mathsf{Sign}(x,m;R)}$ *which on input $m \in \mathbf{Z}/q\mathbf{Z}$ chosen by $\mathcal{A}$, returns a valid signature (r, s) on m where $r := r_x \mod q$ for some fresh $R = (r_x, r_y)$ which was output by $\mathcal{O}^R$ but has not been previously used by $\mathcal{O}^{\mathsf{Sign}}$; else it returns $\perp$.*

*Let $\mathcal{M}$ be the set of queried messages. EC-DSA is enhanced existentially unforgeable under chosen message attack (*e-eu-cma*) if for any such $\mathcal{A}$, the probability $\mathsf{Adv}^{\mathsf{e\text{-}eu\text{-}cma}}_{EC-DSA,\mathcal{A}}$ that $\mathcal{A}$ produces a valid signature on a message $m \notin \mathcal{M}$ is a negligible function of λ.*

Note that $\mathcal{A}$ *chooses* the messages queried to $\mathcal{O}^{\mathsf{Sign}}$, and *knows* (but does not choose) the randomness. Canetti *et al.* [CGG$^+$20] show that in the generic group model EC-DSA is e-eu-cma; and that in some cases, enhanced unforgeability of EC-DSA follows from standard unforgeability of EC-DSA in the random oracle model.

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

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

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

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

We present a game-based definition of security analogous to e-eu-cma: enhanced threshold unforgeability under chosen message attacks (e-tu-cma).

Definition 2 (Enhanced threshold unforgeability). *Consider a (t, n)-threshold EC-DSA protocol $\mathsf{IS} = (\mathsf{IKeyGen}, \mathsf{ISign}, \mathsf{Verif})$, and a PPT algorithm $\mathcal{A}$, having corrupted at most t players, and which is given the view of the protocols* IKeyGen *and* ISign *on input messages of its choice as well as signatures on those messages. As in Definition 1, $\mathcal{A}$ can chose these messages adaptively, and after seeing the randomness used in* ISign*.*

Let $\mathcal{M}$ be the set of aforementioned messages. The protocol IS is enhanced unforgeable if for any such $\mathcal{A}$, the probability $\mathsf{Adv}^{\mathsf{e\text{-}tu\text{-}cma}}_{\mathsf{IS},\mathcal{A}}$ that $\mathcal{A}$ can produce a valid signature on a message $m \notin \mathcal{M}$ is a negligible function of λ.

Adversary model. We consider active (also called malicious/Byzantine) adversaries which are computationally bounded. We also consider two adversarial settings: static adversaries choose the set of corrupted parties in advance, before the interaction begins; whereas adaptive adversaries choose (adaptively) who to corrupt during the course of the protocol.

Identifiable aborts. Our protocols are proven secure even when a majority of players are corrupted, by relying on aborts; i.e. the protocol terminates prematurely if a player is detected as misbehaving. Such protocols are susceptible to "denial of service" attacks, allowing even a single malicious party to force the protocol to abort. A common way to prevent such attacks is to ensure the misbehaving party can be identified upon abort, and accordingly held accountable. We use the definition for secure multi-party computation with identifiable aborts (ID-MPC) introduced by Ishai *et al.* [IOZ14]. This notion ensures that if the computation fails due to an abort, all honest parties learn the identity culprit, i.e. a corrupted party P_i. Concretely, for an arbitrary functionality $\mathcal{F}$, a protocol π realises $\mathcal{F}$ with identifiable abort if π either computes $\mathcal{F}$ or – in case of an abort – outputs the identity P_i of a corrupted player.

2.3 Building blocks from Class Groups

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

We briefly sketch the instantiation given in [CCL+20, Sec. 2.3] and the resulting group generator Gen that we will use in this paper. The interested reader should 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 then consider an element g_q constructed as a random power of $\hat{g}_q$. One can compute an upper bound $\tilde{s}$ for the order of $\hat{g}_q$, using an upper bound of $h(\Delta_K)$ which can be obtained from the analytic class number formula.

For our application q will have at least 256 bits, hence q is prime to $h(\Delta_K)$ except with negligible probability, and so q is prime to the order of $\hat{g}_q$.

Remark 1. Dobson *et al.* [DGS20] recently suggested a new formula to estimate security in a setting where a large set of users (say a billion) share common parameters. In such a setting, public parameters may be targeted by an adversary. They suggest that in this context, an attack with 2^λ running time and a $2^{-\lambda}$ success probability equates to λ bits of security. This leads to estimates on sizes of the discriminant which are much larger than traditionally used. However, this formula is not relevant for our application where there won't be billions of users. Here the standard definition (2^λ running time for a probability of success 1/2 or close to 1) seems more appropriate. As a side note, these estimates in the 'large set of users' context also apply to applications based on an RSA modulus N, as there exists an algorithm which efficiently factors N given the class number of discriminant $-N$ or $-4N$.

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. We further denote $\widehat{G}^q$ the subgroup consisting of all q-th powers in $\widehat{G}$, and it's order $\widehat{s}$. It holds that $\widehat{G}$ is the direct product of $\widehat{G}^q$ and F. We denote $\varpi := \widehat{s}_d$ the group exponent of $\widehat{G}^q$, i.e. the least common multiple of the orders of its elements. Clearly, the order of any element in G^q divides ϖ. In the following the distribution $\mathcal{D}$ from which exponents are sampled is chosen to be close to uniform mod $q \cdot \tilde{s}$, where $\tilde{s}$ is an upper bound for $\widehat{s}$. This means that exponents sampled from $\mathcal{D}$ follow a distribution close to uniform mod q, and mod any divisor of $\widehat{s}$. In particular mod ϖ.

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.

Definition 3 (HSM assumption [CCL+20]). *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:*

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

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

A linearly homomorphic encryption scheme. We recall the linearly homomorphic encryption scheme of [CLT18] whose ind-cpa-security was proven to hold under the HSM assumption. This scheme is the basis of the threshold EC-DSA protocol of Sec. 3. We use the output of $\mathsf{Gen}(1^\lambda, q)$ and as in Def. 3, we set $g_q = \hat{g}_q^t$ for $t \hookleftarrow \mathcal{D}_q$. The public parameters of the scheme are $\mathsf{pp}_{\mathsf{CL}} := (\tilde{s}, f, \hat{g}_q, g_q, \widehat{G}, F, q)$. To instantiate $\mathcal{D}_q$, we set $\tilde{A} \geq \tilde{s} \cdot q \cdot 2^{40}$ s.t. $\{g_q^r, r \hookleftarrow [\tilde{A}]\}$ is at distance less than 2^{-40} from the uniform distribution in G^q. As in [CCL+20], we sample secret keys from a distribution $\mathcal{D}$ s.t. $\{(g_q f)^r, r \hookleftarrow \mathcal{D}\}$ is at distance less than $2^{-\lambda}$ of the uniform distribution in $G = F \times G^q$. The plaintext space is $\mathbf{Z}/q\mathbf{Z}$. The scheme – hereafter refered to as the CL encryption scheme – is depicted in Fig. 1. The

```
Algo. KeyGen(pp_CL)
1. Pick $\mathsf{sk} \hookleftarrow \mathcal{D}$ and $\mathsf{pk} := g_q^{\mathsf{sk}}$
2. Return $(\mathsf{pk}, \mathsf{sk})$

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

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

Fig. 1: Description of the CL encryption scheme

following lemma from [CCL+19] ensures that, in the CL encryption scheme, the distribution followed by the secret keys remains statistically close to uniform mod q even if their value is fixed mod ϖ. The proof can be found in [CCL+19].

Lemma 1. *Let $\mathcal{D}$ be a distribution which is δ-close to $\mathcal{U}(\mathbf{Z}/\widehat{s}q\mathbf{Z})$. For any $x \in G \backslash G^q$, $\pi \leftarrow f^\gamma \in F$ where $\gamma \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ and $k \hookleftarrow \mathcal{D}$, the distributions $\mathcal{D}_1 := \{x, (k \bmod \varpi), \pi \cdot x^k\}$ and $\mathcal{D}_2 := \{x, (k \bmod \varpi), x^k\}$ are 2δ-close.*

We will refer to the above property as the *smoothness* of the CL scheme, as defined in the following definition.

Definition 4 (Smoothness). *The* CL *encryption scheme is said δ_s-smooth if the distribution $\mathcal{D}$ from which secret keys are sampled is $\frac{\delta_s}{2}$-close to $\mathcal{U}(\mathbf{Z}/\widehat{s}q\mathbf{Z})$.*

2.4 Algorithmic assumptions

We here provide further definitions for the algorithmic assumptions on which the security of our protocol relies. As in [CCL+20], we need the HSM assumption (guaranteeing the ind-cpa-security of the linearly homomorphic encryption scheme); the low order assumption (stating that it is hard to find low order elements in the group $\widehat{G}$), and a strong root assumption for class groups (stating that it is hard to find roots in $\widehat{G}$ of random elements of the subgroup $\langle \hat{g}_q \rangle$).

Using similar techniques to [CCL+20], the latter two assumptions allow to significantly improve the efficiency of the ZKAoK needed in our protocol, since whatever the challenge space, if one cannot extract the witness, then one can break at least one of these two assumptions. Consequently they allow to significantly increase the challenge space of our proofs, and reduce the number of rounds in the protocol to achieve a satisfying soundness.

Definition 5 (Low order assumption). *Consider a security parameter $\lambda \in \mathbf{N}$, and $\gamma \in \mathbf{N}$. The* γ-low order problem *(LOP_γ) is $(t(\lambda), \epsilon_{\mathsf{LO}}(\lambda))$-secure for* Gen *if, given the output of* Gen*, no algorithm $\mathcal{A}$ running in time $\leq t(\lambda)$ can output a γ-low order element in $\widehat{G}$ with probability greater than $\epsilon_{\mathsf{LO}}(\lambda)$. More precisely,*

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

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

Remark 2. Belabas *et al.* in [BKSW20] show that if the discriminant belongs to some class of weak primes, then computing small order elements is easy. They also demonstrate that one can easily construct discriminants together with a low order element in their class group without computing the class number. However the likelihood that such a discriminant is chosen at random is negligible. On the other hand, given a discriminant, it seems hard to prove that it is not of such a weak form. Though in our case the discriminant is not prime ($\Delta_K = -q\tilde{q}$), their ideas can be extended to this setting. Hence when relying on the low order assumption, particular attention must be paid in ensuring that discriminants are generated randomly (in our case $\tilde{q}$ must be so), and in particular that special primes are not chosen to meet specific optimization requirements.

We now recall the strong root assumption for class groups.

Definition 6 (Strong root assumption for Class Groups). *Consider a security parameter $\lambda \in \mathbf{N}$, and let $\mathcal{A}$ be a probabilistic algorithm. We run* Gen *on input $(1^\lambda, q)$ to get $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F)$ and we give this output and a random $Y \in \langle \hat{g}_q \rangle$ as an input to $\mathcal{A}$. We say that $\mathcal{A}$ solves the strong root problem for class groups (*SRP*) if $\mathcal{A}$ outputs a positive integer $e \neq 2^k$ for all k and $X \in \widehat{G}$, s.t. $Y = X^e$. In particular, the* SRP *is $(t(\lambda), \epsilon_{\mathsf{SR}}(\lambda))$-secure for* Gen *if any adversary $\mathcal{A}$, running in time $\leq t(\lambda)$, solves the* SRP *with probability at most $\epsilon_{\mathsf{SR}}(\lambda)$.*

3 Threshold EC-DSA protocol

We here present our (t, n)-threshold EC-DSA protocol, divided into five sub-protocols for ease of readability. The first sets up public parameters for the CL encryption scheme, and a common random elliptic curve point $H \in \mathbb{G}$. We then present the key generation sub-protocol for $\mathsf{IKeyGen}$ and a key refresh sub-protocol. Next we provide our protocol for ISign, this is divided into two sub-protocols: an offline protocol Pre-Sign allowing to pre-compute a number of *pre-signatures* before knowing the message to be signed; and an online protocol Sign which takes as input shares of a pre-signature and a message, and outputs a signature. The procedure for identifying misbehaving players is given in Sec. 4.

Interactive Set Up Sub-Protocol. Our protocol requires a set up, as did that of [CCL+20], to ensure that the generator g_q used by all parties in the CL encryption scheme is a *random* generator (this is essential to reduce the smoothness of the ZKAoK we use to the hardness of the strong root assumption in $\widehat{G}$). Our ISetup protocol is that of [CCL+20], with the slight difference that parties also set up a random elliptic curve point H which will be used in the pre-signing protocol. We denote $\mathsf{pp}_{\mathbb{G}} := (\mathbb{G}, P, q)$ the description of the elliptic curve used in EC-DSA. For n parties to collaboratively run ISetup, they proceed as depicted in Fig 2.

P_i	$\mathsf{ISetup}(k, \mathsf{pp}_{\mathbb{G}})$	All players $\{P_j\}_{j \neq i}$
$r_i \xleftarrow{\$} \{0,1\}^k$		
$[\mathsf{c}_i, \mathsf{d}_i] \leftarrow \mathsf{Com}(r_i)$	$\xRightarrow{\mathsf{c}_i}$	
	$\xRightarrow{\mathsf{d}_i}$	$r_i \leftarrow \mathsf{Open}(\mathsf{c}_i, \mathsf{d}_i)$
$\tilde{q} := \mathsf{next\text{-}prime}(\bigoplus_{j=1}^{n} r_j)$		
Compute $\hat{g}_q$ from $q, \tilde{q}$		
$t_i \xleftarrow{\$} [2^{40}\tilde{s}]$ and $g_i \leftarrow \hat{g}_q^{t_i}$		
$h_i \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ and $H_i \leftarrow h_i \cdot P$		
$(\tilde{\mathsf{c}}_i, \tilde{\mathsf{d}}_i) \leftarrow \mathsf{Com}((g_i, H_i))$	$\xRightarrow{\tilde{\mathsf{c}}_i}$	
	$\xRightarrow{\tilde{\mathsf{d}}_i}$	$(g_i, H_i) \leftarrow \mathsf{Open}(\tilde{\mathsf{c}}_i, \tilde{\mathsf{d}}_i)$
$\pi_i := \mathsf{ZKPoK}_{g_i}\{(t_i, h_i) : g_i = \hat{g}_q^{t_i} \wedge H_i = h_i \cdot P\}$	$\xleftrightarrow{\pi_i}$	if a proof fails abort
$g_q \leftarrow \prod_{j=1}^{n} \hat{g}_q^{t_j} = \prod_{j=1}^{n} g_j$		
Erase all data other than $(g_q, H, \mathsf{pp}_{\mathbb{G}})$		

Fig. 2: Threshold CL setup used in $\mathsf{IKeyGen}$

Key Generation Sub-Protocol. After running the ISetup protocol of Fig. 2, all parties possess $(g_q, H, \mathsf{pp}_{\mathbb{G}})$. All parties use this as input for the interactive key generation protocol $\mathsf{IKeyGen}$. Note that in practice ISetup and $\mathsf{IKeyGen}$ would be ran in parallel, and are only here presented separately for ease of readability. As it is exactly the $\mathsf{IKeyGen}$ protocol of [CCL+20], we do not detail the steps of the sub-protocol, but include its' desciption in Fig. 3 for completeness.

Key Refresh Sub-Protocol. This protocol allows players to generate new shares of the EC-DSA secret signing key x and public verification key Q. Each party P_i for $i \in [n]$ runs on input it's previous EC-DSA key shares (u_i, Q_i) satisfying $Q_i = u_i P$; the public parameters $\mathsf{pp}_{\mathbb{G}} = (\mathbb{G}, P, q)$; the verification key Q; and the public parameters $\mathsf{pp}_{\mathsf{CL}}$ for the CL encryption scheme.

P_i	$\mathsf{IKeyGen}(g_q, H, \mathsf{pp}_{\mathbb{G}})$	All players $\{P_j\}_{j \neq i}$
$u_i \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ and $Q_i := u_i P$		
$[\mathsf{kgc}_i, \mathsf{kgd}_i] \leftarrow \mathsf{Com}(Q_i)$		
$(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{CL.KeyGen}(1^\lambda)$	$\xRightarrow{\mathsf{pk}_i \text{ and } \mathsf{kgc}_i}$	
	$\xRightarrow{\mathsf{kgd}_i}$	$Q_i \leftarrow \mathsf{Open}(\mathsf{kgc}_i, \mathsf{kgd}_i)$
Perform (t,n)-VSS share of u_i:		$Q := \sum_{i=1}^n Q_i$
$p_i(X) = u_i + \sum_{k=1}^t a_{i,k} X^k \bmod q$		
Let $\{\sigma_{i,j} := p_i(j)\}_{j \in [n]}$ and $\{V_{i,k} := a_{i,k} P\}_{k \in [t]}$	$\xrightarrow{\text{Send } \sigma_{i,j} \text{ to } P_j}$	
	$\xRightarrow{\{V_{i,k}\}_{k \in [t]}}$	
$\{\sigma_{k,i}\}_k$ 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.		
$\pi_i^{\mathsf{kg}} := \mathsf{ZKPoK}_{X_i}\{(x_i) : X_i = x_i P\}$	$\xleftrightarrow{\pi_{\mathsf{kg},i}}$	
Store (Q, u_i, x_i)		

Fig. 3: Threshold Key Generation

Note that upon key refresh all pre-signatures computed in the previous epoch are erased. This is crucial to ensure that – in our security proof – an adversary can not obtain signatures on two different messages for the same randomness R.

Furthermore, after each execution of Key Refresh, note that the shares u_i^{new} are $(n-1,n)$-additive shares of x. These can be converted into (t,n)-shares x_i of x via. a VSS as in $\mathsf{IKeyGen}$.

Offline Pre-Signature Sub-Protocol. We now present the Pre-Signing sub-protocol which pre-processes signatures before the messages are known. For $i \in [n]$, let x_i denote P_i's (t,n)-share of x output by the $\mathsf{IKeyGen}$ sub-protocol of Fig. 3 (or the output of the latest key refresh, *cf.* Fig. 4); and let $X_i := x_i P$. For each execution a set S of players (satisfying $|S| > t$) is chosen and the secret values $\{w_i\}_{i \in S}$ constitute a (t,t)-aditive secret sharing of the secret signing key x. Each w_i is computed from x_i using Lagrangian coefficients. Furthermore the associated elliptic curve point $W_i := w_i P$ is known to all parties (as W_i can be computed from X_i). The offline Pre-Signing sub-protocol is depicted in Fig. 5.

As in [CCL+20], Phase 2 of the Pre-Signing protocol is a peer-to-peer sub-protocol between each pair of players P_i and P_j, for $i, j \in S, j \neq i$. For private shares $\gamma_i, w_i \in \mathbf{Z}/q\mathbf{Z}$ owned by P_i and $k_j \in \mathbf{Z}/q\mathbf{Z}$ owned by P_j, it allows to convert multiplicative shares $k_j\gamma_i$ and $k_j w_i$ into additive shares $\alpha_{j,i}, \beta_{j,i}, \mu_{j,i}, \nu_{j,i} \in \mathbf{Z}/q\mathbf{Z}$ satisfying $\alpha_{j,i} + \beta_{j,i} = k_j\gamma_i \bmod q$ and $\mu_{j,i} + \nu_{j,i} = k_j w_i \bmod q$. Since a check on values $\mu_{j,i}, \nu_{j,i}$ ensures they are consistent with P_i's secret key share w_i, the sub-protocol computing $\mu_{j,i}, \nu_{j,i}$ is referred to as MtAwc (Multiplicative to Additive with check), whereas that computing $\alpha_{j,i}, \beta_{j,i}$ is referred to as MtA.

$P_i(u_i, Q_i, Q, \mathsf{pp}_{\mathbb{G}}, \mathsf{pp}_{\mathsf{CL}})$	**Key Refresh**	All players
$(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{CL.KeyGen}(1^\lambda)$	$\xRightarrow{\mathsf{pk}_i}$	
$v_{i,1}, \ldots, v_{i,n} \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ s.t $\sum_j v_{i,j} = 0$		
For $j \in [n]$ let $Q_{i,j} := v_{i,j} \cdot P$		
$\mathbf{Y}_i = \{Q_{i,j}\}_{j \in [n]}$		
Sample $\rho_j \xleftarrow{\$} [\tilde{A}]$		
$C_{i,j} := \mathsf{Enc}(\mathsf{pk}_j, v_{i,j}; \rho_j)$	$\xRightarrow{\{C_{i,j}\}_{j\in[n]}, \mathbf{Y}_i}$	if $\sum_{j \in [n]} Q_{i,j} \neq 0_{\mathbb{G}}$
$\pi^{\mathsf{kr}}_{i,j} := \mathsf{ZKPoK}_{C_{i,j}, Q_{i,j}, P}\{(v_{i,j}, \rho_j) :$		then abort
$C_{i,j} = \mathsf{Enc}(\mathsf{pk}_j, v_{i,j}; \rho_j) \wedge Q_{i,j} = v_{i,j} \cdot P\}$	$\xleftrightarrow{\{\pi^{\mathsf{kr}}_{i,j}\}_{j\in[n]}}$	if a proof fails abort
Overwrite old shares with:		
$u_i^{\mathsf{new}} := u_i + \sum_{j \in [n]} \mathsf{Dec}(\mathsf{sk}_i, C_{j,i}) \mod q$		
$Q_i^{\mathsf{new}} := Q_i + \sum_{j \in [n]} Q_{j,i}$		
Erase previously computed pre-signatures		
and all Key Refresh data except $u_i^{\mathsf{new}}, Q, \{\mathsf{pk}_j\}_{j \in [n]}, \mathsf{sk}_i$		

Fig. 4: Key Refresh

Online Signature Sub-Protocol. Our sub-protocol computing signature-shares once the message is known is depicted in Fig. 6. This sub-protocol is executed between the same set S of players that interacted in the Pre-Signing sub-protocol of Fig. 5, all running on input a message m to be signed and a precomputed pre-signature.

4 Identifying Aborts

In this section we show how to identifiy at least one misbehaving player if an abort occurs during an execution of the protocol. We follow a similar idea to that in [CGG+20] and in [CCL+20], adapting their techniques to take into account the fact we use a class group based encryption scheme. Indeed in some specific cases, to identify aborts, we require that parties prove that a ciphertext decrypts to a given value using their decryption key. Hence one proves (in zero-knowledge) knowledge of the secret key allowing to decrypt a ciphertext to a given (public) value. Note that this value may either be the discrete log of an element in F or $\perp$, if decryption fails. The proof we provide is specific to our considered encryption scheme, precisely, it is for the following relation:

$$\mathsf{R}_{\mathsf{Dec}} := \{(\mathsf{pk}, (c_1, c_2), M); \mathsf{sk} \mid c_1, c_2, M \in \widehat{G};\ c_2 \cdot M^{-1} = c_1^{\mathsf{sk}} \wedge \mathsf{pk} = g_q^{\mathsf{sk}}\}.$$

To understand why such a proof is necessary, let us compare Paillier and CL decryptions. With Paillier's cryptosystem, one can extract the encryption randomness from a ciphertext given the decryption key. So if an abort occurs in the protocol, players can publish both the plaintext and encryption randomness underlying ciphertexts which were encrypted using their public key. They can thereby convince other players that the ciphertext is an encryption of the

$P_i(w_i, Q, \mathsf{pp}_{\mathbb{G}}, \mathsf{pp}_{\mathsf{CL}}, \mathsf{sk}_i, \{\mathsf{pk}_j\}_{j\in S})$	**Phase 1**	All players $\{P_j\}_{j\neq i}$
$r_i \xleftarrow{\$} [\tilde{A}]$		
$k_i, \gamma_i \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$		
$c_{k_i} \leftarrow \mathsf{Enc}(\mathsf{pk}_i, k_i; r_i)$		
$[\mathsf{c}_i, \mathsf{d}_i] \leftarrow \mathsf{Com}(\gamma_i P)$	$\xRightarrow{\mathsf{c}_i, c_{k_i}}$	
$\pi_i := \mathsf{ZKAoK}_{\mathsf{pk}_i, c_{k_i}}\{(k_i, r_i) :$		
$((\mathsf{pk}_i, c_{k_i}); (k_i, r_i)) \in \mathsf{R}_{\mathsf{Enc}}\}$	$\xleftrightarrow{\pi_i}$	if a proof fails, abort
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 $\mu_{j,i} \cdot P + B_{j,i} \neq k_j \cdot W_i$ 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 $\{P_j\}_{j\neq i}$
	$\xRightarrow{\delta_i}$	$\delta = \sum_{i\in S} \delta_i = k\gamma$
$\ell_i \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$		
$T_i = \sigma_i \cdot P + \ell_i \cdot H$	$\xRightarrow{T_i}$	
$\widetilde{\pi}_i := \mathsf{ZKPoK}_{T_i}\{(\sigma_i, \ell_i) : T_i = \sigma_i \cdot P + \ell_i \cdot H \in \mathbb{G}\}$	$\xleftrightarrow{\widetilde{\pi}_i}$	if a proof fails, abort
P_i	**Phase 4**	All players $\{P_j\}_{j\neq i}$
	$\xRightarrow{\mathsf{d}_i}$	$\Gamma_i := \mathsf{Open}(\mathsf{c}_i, \mathsf{d}_i)$
$\pi_i^\gamma = \mathsf{ZKPoK}_{\Gamma_i}\{\gamma_i : \Gamma_i = \gamma_i \cdot P\}$	$\xleftrightarrow{\pi_i^\gamma}$	if a proof fails, abort
		$R := \delta^{-1}(\sum_{i\in S} \Gamma_i)$
		Let $R = (r_x, r_y)$ and $r := r_x \bmod q$
P_i	**Phase 5**	All players $\{P_j\}_{j\neq i}$
$\bar{R}_i = k_i \cdot R$	$\xRightarrow{\bar{R}_i}$	
$\pi_i' = \mathsf{ZKPoK}_{\mathsf{pk}_i, c_{k_i}, \bar{R}_i, R}\big\{(k_i, r_i) :$		
$c_{k_i} = \mathsf{Enc}(\mathsf{pk}_i, k_i; r_i) \wedge \bar{R}_i = k_i \cdot R\big\}$	$\xleftrightarrow{\pi_i'}$	if a proof fails, abort
		if $P \neq \sum_{i\in S} \bar{R}_i$ abort
Erase all data except for (ℓ_i, k_i, σ_i) and (Q, u_i, x_i)		
P_i	**Phase 6**	All players $\{P_j\}_{j\neq i}$
$S_i = \sigma_i \cdot R$		
$\pi_i'' = \mathsf{ZKPoK}_{S_i, T_i, R}\{(\sigma_i, \ell_i) : T_i = \sigma_i \cdot P + \ell_i \cdot H \wedge S_i = \sigma_i \cdot R\}$	$\xRightarrow{S_i}$	
	$\xleftrightarrow{\pi_i''}$	if a proof fails, abort
		if $Q \neq \sum_{i\in S} S_i$ abort
Erase all data except for:		
(Q, u_i, x_i) and pre-signature share (R, k_i, σ_i)		

Fig. 5: Pre-Sign: Offline Threshold Pre-Signature Protocol

P_i	**Phase 7**	All players $\{P_j\}_{j \neq i}$
$s_i := mk_i + r\sigma_i$		
	$\xRightarrow{s_i}$	$s := \sum_{i \in S} s_i$,
Erase (R, k_i, σ_i)		if (r, s) is not a valid signature, abort, else return (r, s).

Fig. 6: Sign: Online Threshold Signature Protocol

announced plaintext by re-encryption. However, due of the unknown order of G^q, in CL it is not possible to efficiently compute the encryption randomness, even knowing the decryption key. Consequently parties must prove ciphertexts decrypt to a given value (this may be $\perp$ if decryption fails) in another way.

As the authors do in [CGG+20], we suppose that all the messages are signed from the sender and as a result all the players know the identity of the sender when a message is published. We also consider a local timeout in case of delays in sending. After this timeout a player which has not sent the requested message for a specific phase is considered corrupted. We can list all the possible situations in which the protocol ends with an abort, giving the corresponding solution to identify a misbehaving player. Of course, there can be more than one cheater player, however the aim of identifiable aborts is to detect at least one of them.

Setup. In the set up phase, an abort occurs if a player refuses to decommit, if a proof fails or if a signature on a published value fails. In each case all the players know who the faulty player is, since commitments and proofs are signed.

Key Generation. Aborts in key generation may occur due to a player refusing to decommit, or a proof π_i^{kg} failing to verify. For such types of abort, the culprit is immediately identified. Aborts may also occur if a player complains that the share it received from a Feldman-VSS is inconsistent (i.e. does not verify correctly). In this case the player raising the complaint can publish the received private share and all players can check consistency. If the check passes and the sender's signature is valid, the misbehaving player is the receiver, otherwise it is the sender. After the misbehaving player is identified, the key generation protocol is re-ran with fresh randomness to establish a secure key.

Key Refresh. In a key refresh, an aborts occurs if the sum of the points received from a player is not equal to $0_{\mathbb{G}}$ or a proof $\pi_{i,j}^{\mathsf{kr}}$ fails to verify. In both cases, the sender is detected as the misbehaving player because the proofs or the points are signed.

Pre-Signing and Signing. These protocols may abort for various reasons. When a proof fails or some player refuses to decommit, others can immediately detect the cheater. However, in some cases it is not clear who led the protocol to an end. In these specific cases players have to publish private data used in the computation. We will see that these published values do not reveal information about secret signature key shares. We hereafter list the problematic reasons these protocols may abort.

Problematic types of abort in Pre-Sign and Sign:

1. Phase 2. If a player cannot decrypt the message received (α or μ).
2. Phase 2. If the check on μ fails.
3. Phase 5. If $P \neq \sum \bar{R}_i$.
4. Phase 6. If $Q \neq \sum S_i$.
5. Phase 7. If the signature (r, s) is not valid for the message m.

For an abort of type 5, if the protocol reached this point, then $P = \sum \bar{R}_i$ and $Q = \sum S_i$. Furthermore, all the players know the shares $s_i = mk_i + r\sigma_i \mod q$ of the signature. To detect the cheater, it suffices to verify for which index $s_i \cdot R \neq m \cdot \bar{R}_i + r \cdot S_i$. For the remaining four types of abort a more involved discussion is required.

For aborts occurring in the Pre-Sign protocol, parties may be required to publish their private shares k_i, γ_i and other values depending on the considered case. We emphasize that this does not compromise the secret signing key. Indeed, these aborts occur before revealing the shares s_i of the signature, and as a result publishing k_i or γ_i does not give information on s or s_i, since they are not revealed yet. With this observation, we can present the strategy used to detect cheaters in these remaining cases.
Abort of type 1: Assume P_j is complaining that a ciphertext it received from P_i does not decrypt. Then P_j publishes the faulty ciphertext $\vec{c} := (c_1, c_2)$ along with P_i's signature. Party P_j also reveals $M := c_2 \cdot c_1^{-\mathsf{sk}_j}$, and proves that $(\mathsf{pk}_j, (c_1, c_2), M) \in \mathsf{R}_{\mathsf{Dec}}$, using the proof of Appendix III. If this proof fails, or if $M \in F$, then P_j is lying. Otherwise all parties are convinced that P_i is the cheater.

Abort of type 2: Assume P_j is complaining that $\mu_{j,i} \cdot P + B_{j,i} \neq k_j W_i$. Then P_j publishes:

1. k_j and proves that $(\mathsf{pk}_j, c_{k_j} = (c_{k_j,1}, c_{k_j,2}), f^{k_j}) \in \mathsf{R}_{\mathsf{Dec}}$, so that all the parties can check that k_j is the right decryption of c_{k_j}.
2. $\mu_{j,i}$, the ciphertext $\vec{c}_{k_j w_i} := (c_1, c_2)$ and the elliptic curve point $B_{j,i}$ along with P_i's signatures on the latter two elements. Party P_j also proves that $(\mathsf{pk}_j, (c_1, c_2), f^{\mu_{j,i}}) \in \mathsf{R}_{\mathsf{Dec}}$, using the proof of Appendix III. If this proof fails then P_j is lying. Otherwise all parties are convinced that ciphertext $\vec{c}_{k_j w_i}$ received by P_j decrypts to $\mu_{j,i}$ using P_j's decryption key.

All parties now check that $\mu_{j,i} \cdot P + B_{j,i} \neq k_j W_i$; if so P_i is identified as the cheater, else it is P_j.

Abort of type 3 and 4: Since one can not identify the cheater directly from $P \neq \sum \bar{R}_i$ or from $Q \neq \sum S_i$, in these cases identifying a misbehaving player requires a more convoluted list of actions. To detect who misbehaved, let us consider the protocol steps leading up to an abort of type 3. Note that aborts of type 3 and 4 occur after proofs $\{\pi_{\mathsf{kg},i}\}_{i\in[n]}$ and $\{\pi_i, \widetilde{\pi}_i, \pi_i^{\gamma}, \pi_i'\}_{i\in S}$ have been accepted.

The identification protocol uses new techniques, and requires the use of a zero-knowledge proof for $\mathsf{R}_{\mathsf{Dec}}$ to prove ciphertexts decrypt to a given value (e.g. the proof of Appendix III). In order to prove that the players indeed ran the protocol correctly, it is necessary and sufficient to prove that for $i \in S$ all the following consistency items hold:

(i) The value k_i input to the MtA protocol is consistent with that input to the MtAwc protocol. This holds unconditionally since only a single encryption of k_i is broadcast in Phase 1.
(ii) The value w_i input to the MtAwc protocol is consistent with the public value $W_i = w_i \cdot P$ that is associated with player P_i. Under the soundness of the Schnorr ZKPoK used for π_i^{kg} in $\mathsf{IKeyGen}$, this item holds.
(iii) The value $\tilde{\gamma}_i$ input to the MtA protocol is consistent with $\mathit{\Gamma}_i = \gamma_i \cdot P$ that is decommitted to in Phase 4, i.e. $\gamma_i = \tilde{\gamma}_i$.
(iv) The value δ_i published in Phase 3 is consistent with the shares received during the MtA protocol. In particular, the following should hold:

$$\delta_i = k_i\gamma_i + \sum_{j\neq i}\alpha_{ij} + \sum_{j\neq i}\beta_{ji}.$$

(v) The value S_i published in Phase 6 is consistent with the shares received during the MtAwc protocol. In particular, it must hold that

$$S_i = \sigma_i \cdot R \quad \text{and} \quad \sigma_i = k_i w_i + \sum_{j\neq i}\mu_{ij} + \sum_{j\neq i}\nu_{ji}.$$

We now distinguish aborts of type 3 and 4.

Identification - Abort of type 3 in Phase 5. For this to occur, either consistency item (iii) or (iv) does not hold (since S_i has not been computed yet, item (v) is not relevant here). We hereafter explain how, for both of these inconsistency types, parties can identify a cheating player.

(iii) Each party P_j publishes (in order):
  (a) For $i \neq j$, $\beta_{i,j}$ and γ_j. All players then check that $\mathit{\Gamma}_j$ revealed in Phase 4 satisfies $\mathit{\Gamma}_j = \gamma_j \cdot P$. If the check fails, P_j is identified as a cheater.
  (b) After all $\beta_{\ell,i}$ and γ_i, for $i \neq j$ and for $\ell \neq i$ have been published, P_j reveals k_j and performs a ZK proof that $(\mathsf{pk}_j, c_{k_j}, f^{k_j}) \in \mathsf{R}_{\mathsf{Dec}}$. If the proof is accepted, all parties are convinced c_{k_j} decrypts to k_j using P_j's decryption key; else P_j is identified as a cheater.
  (c) For $i \neq j$, reveal $\alpha_{j,i}$ and $c_{k_j\gamma_i}$. Then perform a ZK proof that $(\mathsf{pk}_j, c_{k_j\gamma_i}, f^{\alpha_{j,i}}) \in \mathsf{R}_{\mathsf{Dec}}$. If the proof is accepted, all parties are convinced $c_{k_j\gamma_i}$ decrypts to $\alpha_{j,i}$ using P_j's decryption key; else P_j is identified as a cheater.

  Then for $i \neq j$, all players can compute:

$\widetilde{\beta}_{i,j} := k_i\gamma_j - \alpha_{i,j}$ and check if $\beta_{i,j} = \widetilde{\beta}_{i,j}$. If it is not true for some index i, then P_j is the cheater (for using $\tilde{\gamma}_j \neq \gamma_j$ in the MtA protocol).
We now argue that if all such checks pass, item (iii) holds, i.e., all players are convinced that $\tilde{\gamma}_j \neq \gamma_j$. Observe that if P_j used a different value $\tilde{\gamma}_j$ in the MtA than the γ_j published in step (a) satisfying $\Gamma_j = \gamma_j \cdot P$, then to have $\beta_{i,j} = \widetilde{\beta}_{i,j}$, party P_j must have predicted the value $k_i\gamma_j - \alpha_{i,j}$ without knowing $\alpha_{i,j}$ or k_i. Under the smoothness of the encryption scheme, the distribution followed by k_i is uniformly random in $\mathbf{Z}/q\mathbf{Z}$ from P_j's view in step (a), hence P_j (who only knows $\beta_{i,j} = k_i\tilde{\gamma}_j - \alpha_{i,j}$) cannot predict $\widetilde{\beta}_{i,j} = k_i\gamma_j - \alpha_{i,j}$ with probability significantly greater than $1/q$.

(iv) Now if consistency of (iii) holds, all parties can check that $\delta_j = k_j\gamma_j + \sum_{i\neq j} \alpha_{j,i} + \sum_{i\neq j} \beta_{i,j}$; if not P_j is identified as the cheater. If equality holds then consistency of (iv) holds (i.e. consistency of δ_j).

Identification - Abort of type 4 in Phase 6. Since this failure occurs in Phase 6, we know that consistency of (i), (ii), (iii) and (iv) hold. The abort must hence be due to (v) not holding. To detect a cheater, each P_j does the following:

1. Publish k_j and perform a ZK proof that $(\mathsf{pk}_j, c_{k_j}, f^{k_j}) \in \mathsf{R}_{\mathsf{Dec}}$. If the proof is accepted, all parties are convinced c_{k_j} decrypts to k_j using P_j's decryption key; else P_j is identified as a cheater.
2. For all $i \neq j$, publish $\mu_{j,i}$ and $c_{k_j w_i}$. Perform a ZK proof that $(\mathsf{pk}_j, c_{k_j w_i}, f^{\mu_{j,i}}) \in \mathsf{R}_{\mathsf{Dec}}$. If the proof is accepted, all parties are convinced $c_{k_j w_i}$ decrypts to $\mu_{j,i}$ using P_j's decryption key; else P_j is identified as a cheater.
3. For all $i \neq j$, publish the elliptic curve point $B_{j,i}$ received from P_i (and the signature). Notice that since checks of Phase 2 passed, for all i, j it holds that $\mu_{j,i} \cdot P + B_{j,i} = k_j \cdot W_i$ and so $B_{j,i} = \nu_{j,i} \cdot P$.

If the above checks passed, all parties can compute:

$$\Sigma_j = \sigma_j \cdot P := k_i W_i + \sum_{i\neq j} \mu_{j,i} \cdot P + \sum_{i\neq j} B_{i,j}.$$

Finally, P_j performs a ZK proof that the discrete log in base P of Σ_j is equal to the discrete log in base R of S_j. Precisely, let for public parameters $(\mathbb{G}, P, q)$, let $\mathsf{R}_{\mathsf{log}} := \{(R, S, T) \in \mathbb{G}^3; \sigma \mid R = \sigma \cdot S \wedge T = \sigma \cdot P\}$. Then P_j performs a ZKPoK π_j^{log} that $(R, S_j, \Sigma_j) \in \mathsf{R}_{\mathsf{log}}$; this can be done as described in [CP93]. If this proof fails, P_j is identified as a cheater.

Note that if none of these proofs fail, under the soundness of the aforementioned ZK proofs it holds that, for $i \in S$, consistency of (v) holds, and no aborts occur. This concludes the description of the identification procedure.

5 Security

To prove that our protocol is secure, we demonstrate that if there exists a PPT algorithm $\mathcal{A}$ which breaks enhanced unforgeability of the threshold EC-DSA protocol of Fig. 3, 4, 5 and 6, then one can devise an algorithm $\mathcal{S}$ using $\mathcal{A}$ to break the enhanced unforgeability of centralised EC-DSA. To this end $\mathcal{S}$ simulates the environment of $\mathcal{A}$, so that $\mathcal{A}$'s view of its interactions with $\mathcal{S}$ are indistinguishable from $\mathcal{A}$'s view in a real execution of the protocol.

In Subsections 5.1 and 5.2, we first prove our (t, n)-threshold protocol, for any $t < n$, secure against static corruptions. Next, in Subsection 5.3 we show that if all players participate in the signing algorithm ($t = n - 1$) one can easily adapt the proof for static corruptions to the adaptive case. We note that [CGG$^+$20] also only prove their protocol secure for $t = n - 1$. They state that using techniques of Gennaro et al. [GG18] one can immediately derive a full threshold protocol. We emphasize that in order to build our (t, n)-protocol, for any $t < n$, *we do* use the techniques of [GG18], hence it may be the case that it is also secure against active adversaries. However, as detailed in Subsection 5.3, it remains unclear to us how one can claim security against adaptive adversaries in this setting.

In both adversarial settings, $\mathcal{S}$ gets as input an EC-DSA public key Q, where $Q = x \cdot P$, and can query oracles $\mathcal{O}^R$ and $\mathcal{O}^{\mathsf{Sign}(x,\cdot,\cdot)}$ of Def 1. After this query phase, $\mathcal{S}$ must output a forgery, i.e. a signature s for a message m of its choice, which it did not receive from the oracle.

5.1 Security of the Full Threshold Protocol with Identifiable Aborts against Static Adversaries

As all players play symmetric roles in the protocol, it suffices to demonstrate that if $\mathcal{A}$ corrupts $\{P_j\}_{j>1}$, one can construct $\mathcal{S}$ simulating P_1 s.t. the output distribution of $\mathcal{S}$ is indistinguishable from $\mathcal{A}$'s view in an interaction with an honest party P_1.

Simulating the Key Generation Protocol. On input $Q = x \cdot P$, the forger $\mathcal{S}$ must set up in its simulation with $\mathcal{A}$ this same verification key Q (without knowing x). This will allow $\mathcal{S}$ to subsequently simulate interactively signing messages with $\mathcal{A}$, using the output of its' (enhanced) EC-DSA signing oracle.

The main differences with the proof of [GG20], arises from the fact $\mathcal{S}$ knows it's own decryption key sk_1, but does not extract that of other players. As in [CCL$^+$20], the linearly homomorphic encryption scheme we use results from projective hash functions, whose security is statistical, thus the fact $\mathcal{S}$ uses its' secret key does not compromise security, and we can still reduce the security of the protocol to the smoothness of the CL scheme. However as we do not prove knowledge of secret keys associated to public keys in the key generation protocol, $\mathcal{S}$ can not extract the decryption keys of corrupted players.

Our proof strategy is similar to that of [CCL$^+$20], with the difference that we here take into account the simulation of identifiable aborts; this changes the way one defines semi correct executions. The simulation is described below.

Simulating Key Generation - Description of $\mathcal{S}$:

1. $\mathcal{S}$ 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. $\mathcal{S}$ samples a CL encryption key pair $(\mathsf{pk}_1, \mathsf{sk}_1) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$, and a random value $u_1 \in \mathbf{Z}/q\mathbf{Z}$. It computes $[\mathsf{kgc}_1, \mathsf{kgd}_1] \leftarrow \mathsf{Com}(u_1 P)$ and broadcasts pk_1 and kgc_1. In return, $\mathcal{S}$ receives the public keys $\{\mathsf{pk}_j\}_{j \in [n], j \neq 1}$ and commitments $\{\mathsf{kgc}_j\}_{j \in [n], j \neq 1}$.
4. $\mathcal{S}$ 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.
5. $\mathcal{S}$ 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 $\widehat{Q}_1$.
6. $\mathcal{A}$ will broadcast the decommitments $\{\widehat{\mathsf{kgd}}_j\}_{j \in [n], j \neq 1}$. Let $\{\widehat{Q}_j\}_{j=2 \ldots n}$ be the committed value revealed by $\mathcal{A}$ at this point (this could be $\perp$ if $\mathcal{A}$ refuses to decommit).
7. All players compute the public signing key $\widehat{Q} := \sum_{i=1}^{n} \widehat{Q}_i$. If any $\widehat{Q}_i = \perp$, then $\widehat{Q} := \perp$.
8. 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.
9. $\mathcal{S}$ simulates $\pi_{\mathsf{kg},1}$ (the ZKPoK that it knows x_1 corresponding to X_1). Then, for $j \in [n]$, $j \neq 1$, $\mathcal{S}$ receives from $\mathcal{A}$ a ZKPoK of x_j satisfying $X_j := x_j \cdot P$; from which $\mathcal{S}$ can extract x_j.

Simulating the Key Refresh Protocol. For all honest players P_i for $i \in \mathbf{H}$, $\mathcal{S}$ runs the prescribed steps of the Key Refresh protocol.

Simulating Protocols Pre-Sign and Sign. After the key generation is over, the simulator must handle the signature queries issued by $\mathcal{A}$. Recall that $\mathcal{A}$ can issue two types of queries:

- oracle $\mathcal{O}^R$ to obtain a uniformly random point $R = (r_x, r_y)$ in $\mathbb{G}$:
- oracle $\mathcal{O}^{\mathsf{Sign}(\mathsf{sk}, m; R)}$ which on input a message m chosen by $\mathcal{A}$, returns a valid signature (r, s) for m where $r := r_x \bmod q$ if $R = (r_x, r_y)$ was queried to $\mathcal{O}^R$; else it returns $\perp$.

The simulator simulates P_1 in the threshold signature protocol on input R for the offline phase (Phases 1-6), and a correct signature (r, s) for m under the public key Q for the online phase (Phase 7). We stress that though the simulator knows the decryption key sk_1, and P_1's EC-DSA "public key share" $W_1 = w_1 \cdot P$; it does not know the secret value w_1 associated with P_1. However it does know the shares w_j, $j > 1$ of all other players (extracted from the Schnorr proofs in the Key Generation phase).

The simulation of the Pre-Signing and Signing protocols is based on [GG20] and [CCL+20]. In the following simulation $\mathcal{S}$ aborts whenever the protocol is supposed to abort, i.e., whenever $\mathcal{A}$ refuses to decommit a committed value, a ZK proof fails, a check does not pass or if the signature (r, s) does not verify.

Simulating Pre-signing and Signing - Description of $\mathcal{S}$:

Phase 1: As in a real execution, $\mathcal{S}$ samples $k_1, \gamma_1 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$, $r_1 \xleftarrow{\$} [\widetilde{A}]$ 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 $\mathcal{A}$. $\mathcal{S}$ checks the proofs are valid and extracts the encrypted values $\{k_j\}_{j \in S, j \neq 1}$ from which it computes $k := \sum_{i \in S} k_i$.

Phase 2: Recall that during the regular run of the protocol, P_1 will engage in two MtA protocols and two MtAwc protocols with each other player $P_j, j \in S, j \neq 1$. $\mathcal{S}$ runs the protocol for P_1 as follows:

(a) *Initiator for MtA with k_1 and γ_j*: Since $\mathcal{S}$ knows k_1, it runs the protocol as would an honest P_1; it also decrypts $c_{k_1 \gamma_j}$ received from P_j thereby obtaining $\alpha_{1,j} \bmod q$.

(b) *Respondent for MtA with k_j and γ_1*: Since $\mathcal{S}$ knows γ_1, it runs the protocol as would an honest P_1. Recall that $\mathcal{S}$ extracted k_j from π_j in Phase 1, it also knows $\beta_{j,1}$ (as $\mathcal{S}$ chose it), hence $\mathcal{S}$ can compute P_j 's share $\alpha_{j,1} := k_j \gamma_1 - \beta_{j,1} \bmod q$.

(c) *Initiator for MtAwc with k_1 and w_j (note that the first message sent in this sub-protocol is common to all players in both MtA (item (a)) and MtAwc (item (c)))*: Since $\mathcal{S}$ knows k_1, it runs the protocol as would an honest P_1; decrypting $c_{k_1 w_j}$ to obtain $\mu_{1,j}$; and checking that $\mu_{1,j} P + B_{1,j} = k_1 W_j$. Furthermore, recall that $\mathcal{S}$ extracted x_j from $\pi_{\mathsf{kg},j}$ in KeyGen, hence it knows w_j, so $\mathcal{S}$ can compute $\nu_{1,j} = k_1 w_j - \mu_{1,j} \bmod q$.

(d) *Respondent for MtAwc with k_j and w_1*: Here, $\mathcal{S}$ only knows $W_1 = w_1 \cdot P$ (but not w_1). So it samples a random $\mu_{j,1} \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ and sets $c_{k_j w_1} \leftarrow \mathsf{Enc}(\mathsf{pk}_j, \mu_{j,1})$, and $B_{j,1} := k_j \cdot W_1 - \mu_{j,1} \cdot P$. Finally $\mathcal{S}$ sends the cipertexts and the point.

Note that at this point $\mathcal{S}$ knows:

- k_i for each $i \in S$, w_j for each $j \in S, j \neq 1$
- $\alpha_{1,j}, j \in S, j \neq 1$ as initiator for MtA, $\alpha_{j,1}, \beta_{j,1}, j \in S, j \neq 1$ as respondent for MtA. Indeed, $\alpha_{1,j}$ is a value decrypted by $\mathcal{S}$; $\beta_{j,1}$ is chosen by $\mathcal{S}$; and $\alpha_{j,1} = k_j \gamma_1 - \beta_{j,1}$, where $\mathcal{S}$ knows all the values on the right, and so can compute $\alpha_{j,1}$.
- $\mu_{1,j}, \nu_{1,j}, j \in S, j \neq 1$ as initiator for MtAwc, $\mu_{j,1}, \nu_{j,1}, j \in S, j \neq 1$ as respondent for MtAwc. Indeed, $\mu_{1,j}$ is a value decrypted by $\mathcal{S}$ decrypts; $\nu_{j,1}$ is chosen by $\mathcal{S}$; $\mu_{j,1}$ is chosen by $\mathcal{S}$ in (d); $\nu_{1,j} = k_1 w_j - \mu_{1,j}$, where $\mathcal{S}$ knows all the values on the right.

After all these sub-protocols, $\mathcal{S}$ computes $\delta_1 := k_1 \gamma_1 + \sum_{j \in S, j \neq 1} \alpha_{1,j} + \sum_{j \in S, j \neq 1} \beta_{j,1}$. Note that $\mathcal{S}$ does not know the internal values from the MtA and MtAwc protocols executed by two players that are both controlled by the adversary. Hence $\mathcal{S}$ is not able to compute the values σ_j and δ_j for $j \in S, j \neq 1$. Furthermore $\mathcal{S}$ cannot compute σ_1 since it doesn't know the value w_1, but it can compute

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

since it knows all the values $\{k_j\}_{j\in S}$, $\{w_j\}_{j\in S, j\neq 1}$, it chooses the random values $\mu_{i,1}$ and it can compute all of the shares $\nu_{1,j} = k_1 w_j - \mu_{1,j} \mod q$. Furthermore, as $\mathcal{S}$ knows W_1, it can compute $\Sigma_1 := \sigma_1 \cdot P = k_1 W_1 + \sum_{j\neq 1}(\mu_{1,j} + \nu_{j,1})P$.

Phase 3: $\mathcal{S}$ broadcasts δ_1 and receives all the $\{\delta_j\}_{j\in S, j\neq 1}$ from $\mathcal{A}$. Let $\widetilde{\delta} := \sum_{i\in S} \delta_i$. Next $\mathcal{S}$ samples a random $\ell_1 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ and broadcasts $T_1 := \Sigma_1 + \ell_1 \cdot H$. Note that $\mathcal{S}$ does not know σ_1, so it simulates the ZK proof $\tilde{\pi}_1$. Then, from the proofs $\tilde{\pi}_j$ received from $\mathcal{A}$, $\mathcal{S}$ extracts the values $(\widehat{\sigma}_j, \ell_j)$ for $j \in S, j \neq 1$. We hereafter denote $\widehat{\sigma}_C := \sum_{j\in S, j\neq 1} \widehat{\sigma}_j$.

Phase 4: $\mathcal{S}$ broadcasts d_1 which decommits to Γ_1, and $\mathcal{A}$ reveals $\{\mathsf{d}_j\}_{j\in S, j\neq 1}$ which decommit to $\{\Gamma_j\}_{j\in S, j\neq 1}$. From the proofs π_j^γ on Γ_j, $\mathcal{S}$ can extract γ_j for each $j \in S, j \neq 1$; these are consistent with the values used in Phase 1 thanks to the binding property of the commitment scheme. Now $\mathcal{S}$ can compute

$$\delta = (\sum_{i\in S} k_i) \cdot (\sum_{i\in S} \gamma_i) = k\gamma,$$

where $\gamma = \sum_{i\in S} \gamma_i$. Note that $\mathcal{A}$ may have used different values $\widetilde{\gamma}_i$ in the MtA protocol than the γ_i committed to in Phase 1, hence we denote them with a tilde.

At this point $\mathcal{S}$ can detect if the values published so far by $\mathcal{A}$ are consistent; note that $\mathcal{S}$ will behave differently in Phases 5, 6 and 7 depending on this detection.

To detect inconsistencies, $\mathcal{S}$ first computes $\widetilde{R} := \widetilde{\delta}^{-1}(\sum_i \Gamma_i)$.

Then using the values k_i extracted in Phase 1, $\mathcal{S}$ checks that $\sum_i k_i \cdot \widetilde{R} = P$. Notice that if

$$\sum_i k_i \cdot \widetilde{R} = k \cdot \widetilde{R} = k\widetilde{\delta}^{-1} \sum_{i\in S} \gamma_i \cdot P = k\widetilde{\delta}^{-1}\gamma \cdot P = P \Rightarrow \widetilde{\delta} = k\gamma = \delta$$

The simulator can also detect if the values σ_j computed in Phase 2 are consistent with those used to compute points T_j in Phase 3; in particular $\mathcal{S}$ checks that $\widehat{\sigma}_C = \sigma_C$. We thus distinguish two types of executions: an execution is said to be *semi-correct* if

$$\sum_i k_i \widetilde{R} = P \qquad \text{and} \qquad \widehat{\sigma}_C = \sigma_C$$

which, as explained above, implies that $\delta = \widetilde{\delta}$ and $\widehat{\sigma}_C = \sigma_C$. If either of the above equalities do *not* hold, the execution is said to be *non semi-correct.* Now $\mathcal{S}$ adapts its behaviour depending on the type of execution:

- **Semi-correct execution:**
  1. $\mathcal{S}$ invokes oracle $\mathbb{O}^R$ to obtain $R = (r_x, r_y)$.
  2. $\mathcal{S}$ sets $\widehat{\Gamma}_1 := \widetilde{\delta} \cdot R - \sum_{i \in S, i \neq 1} \Gamma_i$, so that $R = \widetilde{\delta}^{-1} \left(\widehat{\Gamma}_1 + \sum_{i \in S, i \neq 1} \Gamma_i \right)$. Then $\mathcal{S}$ rewinds $\mathcal{A}$ to the decommitment step in Phase 4, equivocates P_1's commitment so that it decommits to $\widehat{\Gamma}_1$ instead of Γ_1.
- **Non semi-correct execution:** $\mathcal{S}$ simply moves on to Phase 5.

Phase 5:
- **Semi-correct execution:** $\mathcal{S}$ publishes $\bar{R}_1 := P - \sum_{i \in S, i \neq 1} k_i \cdot R$ together with π'_1: a simulated ZKP of consistency with $c_{k_1} = \mathsf{Enc}_1(k_1)$ (note that in this case $\bar{R}_1 \neq k_1 \cdot R$ due to the rewinding).
- **Non semi-correct execution:** $\mathcal{S}$ publishes $\bar{R}_1 := k_1 \cdot R$ together with π'_1: a real ZKP of consistency with $c_{k_1} = \mathsf{Enc}_1(k_1)$ (this proof needn't be simulated).

Phase 6:
- **Semi-correct execution:** $\mathcal{S}$ publishes $S_1 := Q - \sum_{j \in S, j \neq 1} \sigma_j R$ together with π''_1: a simulated ZKP of consistency with T_1 (again in this case the simulated $S_1 \neq \sigma_1 \cdot R$ due to the rewinding).
- **Non semi-correct execution:** $\mathcal{S}$ has P_1 publish $S_1 := \sigma_1 R$ together with π''_1: a real ZKP of consistency with T_1 (this proof needn't be simulated).

In a non semi-correct execution, at least one of the the adversary's proofs π'_j or π''_j for some $j > 1$ will fail, and the protocol will abort.

Phase 7: $\mathcal{S}$ invokes the second oracle $\mathbb{O}^{\mathsf{Sign}(\mathsf{sk}, m; R)}$ with input m. In return, $\mathcal{S}$ receives the valid signature (r, s) on m, where $r = r_x \bmod q$, for some $R = (r_x, r_y)$ computed in a previous offline phase (in particular in one that was semi-correct, since it concluded successfully).
Now $\mathcal{S}$ knows $s_C = \sum_{j \in S, j \neq 1} s_j$ because $s_C = m \sum_{j \in S, j \neq 1} k_j + \sigma_C r$ where σ_C is as defined in the simulation of Phase 2 (Note: if $\mathcal{A}$ cheats in Phase 7 – denoting $\{\tilde{s}_i\}_{i>1}$ the values $\mathcal{S}$ receives from $\mathcal{A}$ in Phase 7, and $\tilde{s}_C := \sum_{i>1} \tilde{s}_i$ – it is possible that $s_C \neq \tilde{s}_C$). So $\mathcal{S}$ computes the share s_1 consistent with (r, s) and s_C as $s_1 := s - s_C$. Finally, $\mathcal{S}$ broadcasts this value s_1.

Simulating Identification.

Simulating Identification of aborts in Key Generation. If an abort occurs in the Key Generation protocol, $\mathcal{S}$ runs the identification protocol as would P_1 in a real execution. Furthermore, if some player P raises a compliant against P_1 (simulated by $\mathcal{S}$), then P is detected as a cheater since the simulation of Feldman VSS is done in such a way that corrupted players receive values which pass the verification check.

Simulating Identification of aborts in Pre-Sign and Sign. For any of the trivial types of abort allowing to immediately detect the faulty player, $\mathcal{S}$ has nothing to simulate. Consider the problematic types of abort listed on page 16:

- Abort of type 1: if $\mathcal{S}$ cannot decrypt another player's ciphertext, since $\mathcal{S}$ knows sk_1, it can run the proof for relation R_{Dec} as would P_1.
- Abort of type 2: if $\mathcal{S}$ announces that the check on $\mu_{1,j}$ fails (for some $j > 1$), it runs the identification protocol as would P_1. Conversely, if some player P_j for $j > 1$ complains about the $\mu_{j,1}$ it received, observe that: if $\mu_{j,1}$ is the real decryption of $c_{k_j w_1}$ (which it must be if the proof for R_{Dec} provided by P_j is valid), then since the point $B_{j,1}$ sent by $\mathcal{S}$ to P_j was computed as $B_{j,1} := k_j \cdot W_1 - \mu_{j,1}$, necessarily the equality test will pass. Observe that the value $\nu_{j,1}$ is not revealed; hence no other (corrupted) party can check that $\mathcal{S}$ knows $\nu_{j,1}$ such that $B_{j,1} = \nu_{j,1} \cdot P$. Hence the simulation ends correctly.
- Abort of type 3: $\mathcal{S}$ follows the real identification procedure (*cf.* page 17).
- Abort of type 4: $\mathcal{S}$ follows the real identification procedure, up until it needs to prove knowledge of σ_1 satisfying $S_1 = \sigma_1 \cdot R$. Since $\mathcal{S}$ does not know σ_1, it simulates the proof π_1^{log}.
- Abort of type 5: here an abort occurs if the computed signature (r, s) is not valid, as no extra values need to be published to identify the cheater, $\mathcal{S}$ has nothing to simulate.

  Let us denote $\{s_i\}_{i>1}$ the values computed during the pre-sign protocol (which are correct since no abort occurred), and let us denote $\{\tilde{s}_i\}_{i>1}$ the values that $\mathcal{A}$ broadcasts in Phase 7. Let us further denote $s_C := \sum_{i>1} s_i$ and $\tilde{s}_C := \sum_{i>1} \tilde{s}_i$. Now observe that – as long as $s_C = \tilde{s_C}$ – from the way $\mathcal{S}$ computes s_1 (i.e. $s_1 := s - s_C$), since the signature it receives from its oracle is valid for verification key Q, the computed signature is necessarily valid. Conversely, if $s_C \neq \tilde{s_C}$, $\mathcal{S}$ sets $s_1 := s - s_C$ and aborts. To identify the cheater, everyone checks $s_i \cdot R = m \cdot \bar{R}_i + r \cdot S_i$ for all i. From the way $\mathcal{S}$ computed $s_1 R$, it will not be identified as the cheater.

5.2 Indistinguishability of Real and Simulated Environments

We here argue that a static adversary $\mathcal{A}$ can not distinguish a real execution of the protocol – interacting with P_1 – from a simulated execution. We distinguish semi-correct and non semi-correct executions.

Semi-Correct Executions. Lemma 2 states the assumptions under which indistinguishability holds. Regarding the key generation, pre-signing and signing sub-protocols, the proof resembles that of [CCL+20, Lem. 4], and is deferred to Appx. IV. Furthermore the simulator runs the key refresh sub-protocol as in a real execution, hence the simulation there is perfect. We here only detail the indistinguishability of identification procedures.

Lemma 2. *Assuming the strong root and C-low order assumptions hold for Gen; the CL encryption scheme is δ_s-smooth; 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.*

Indistinguishability of identification procedures. As demonstrated in Appx. IV, except with negligible probability, a simulated execution results in an abort if and only if a real execution would (this must be true for real and simulated views of the adversary to be indistinguishable). Of course if no abort occurs, the simulation of the identification procedure is not an issue. Now assuming there is an abort, consider the problematic types of abort listed on page 16. For an abort of type 5, identifying the culprit is trivial, and there is no impact on the view $\mathcal{A}$ has of real and simulated executions. If an abort of type 1 or 2 occurs, in all of our considered game steps $\mathcal{S}$ can honestly perform the proof for relation R_{Dec} as would P_1, hence $\mathcal{A}$'s view of these identification procedures is identical in all game steps, and therefore in real and simulated executions.

Finally, as we are here considering the simulation of semi-correct executions aborts of type 3 and 4 do not occur (indeed, the occurrence of such aborts means we are in a non-semi-correct executions). Hence any abort which may occur in a semi-correct execution is perfectly simulated.

Non Semi-Correct Executions.

Lemma 3. *If the strong root and C-low order assumptions hold for* Gen *then the simulation is computationally indistinguishable from a non-semi-correct real execution.*

Proof. In this case both real and simulated executions of the protocol abort before Phase 7; so in all situations where semi-correct and non semi-correct simulations differ, in a non semi-correct execution $\mathcal{S}$ follows the protocol as would P_1; hence the non semi-correct simulation of the pre-signing sub-protocol is indistinguishable from a non semi-correct real execution.
Identification procedures. Here aborts are either of type 3 or 4. For type 3, $\mathcal{S}$ follows the real identification procedure, hence the simulation is perfect. For type 4 the only difference is that $\mathcal{S}$ simulates π_j^{log}, so $\mathcal{A}$'s view is indistinguishable.

Concluding the proof. The forger $\mathcal{S}$ simulating $\mathcal{A}$'s environment can detect whether we are in a semi-correct execution or not. Consequently $\mathcal{S}$ always knows how to simulate $\mathcal{A}$'s view and all simulations are indistinguishable from real executions of the protocol. Moreover if $\mathcal{A}$, having corrupted up to t parties in the threshold EC-DSA protocol, outputs a forgery, since $\mathcal{S}$ set up with $\mathcal{A}$ the same public key Q it received from its' EC-DSA challenger, and randomness R it received from $\mathcal{O}^R$, $\mathcal{S}$ can use this signature as its own forgery, thus breaking the enhanced existential unforgeability of centralised EC-DSA. Hence the following theorem, which captures the protocol's security, follows from Lemmas 2 and 3.

Theorem 1. *Assuming EC-DSA is enhanced existentially unforgeable under chosen message attacks; the strong root and C-low order assumptions hold for* Gen*; the* CL *encryption scheme is* $\mathsf{ind\text{-}cpa}$*-secure; and the commitment scheme is non-malleable and equivocable, it holds that the (t, n)-threshold EC-DSA protocol of Fig. 3-4-5-6 is enhanced existentially unforgeable against static adversaries.*

5.3 Security Against Adaptive Adversaries

Our protocol can further be proved secure against adaptive corruptions in the specific case $t = n - 1$, i.e. all parties must participate in the signing phases (we leave the study of adaptive security for any $t \leq n - 1$ for future work).

Theorem 2. *Assuming EC-DSA is* e-eu-cma*; the* DL *assumption holds in* $\mathbb{G}$*; the strong root and* C*-low order assumptions hold for* Gen*; the* CL *encryption scheme is* ind-cpa*-secure; and the commitment scheme is non-malleable and equivocable, then the* $(n-1, n)$*-threshold EC-DSA protocol of Fig. 7-4-5-6 is* e-tu-cma *against adaptive corruptions.*

As explained hereafter, in the specific case $t = n-1$, the security proof very much resembles that against static adversaries, hence details are deferred to Appx. V.

Proof strategy. In the context of adaptive corruptions the adversary $\mathcal{A}$ can choose to corrupt players throughout the execution of the protocol. When such a corruption occurs, $\mathcal{A}$ is given the corrupted party's internal state: $\mathcal{A}$ learns the party's secret values, randomness, and any other information the party may have stored from previous interactions. Hence to ensure $\mathcal{A}$ is unable to distinguish between real and simulated executions, one must ensure that $\mathcal{A}$ can not detect any inconsistencies when it chooses to corrupt a new party P. If $\mathcal{A}$ corrupts a player which *does* reveal inconsistencies, the simulator $\mathcal{S}$ rewinds the protocol. One should thus minimise the number of players possessing inconsistent values, so as to reduce the number of rewinds. In particular, if a player P is simulated as an honest player following the real protocol, then it can only give consistent values to $\mathcal{A}$ if it is corrupted. A crucial point of using the CL encryption scheme is that $\mathcal{S}$ knows the decryption keys of honest players; so if an honest player is corrupted, $\mathcal{S}$ can give this secret key to $\mathcal{A}$, which is consistent with the encryption key. This is not immediate in a situation where the secret key is not known by $\mathcal{S}$. In our proof, $\mathcal{S}$ simulates the behaviour of each honest player, revealing the relevant internal states upon corruption; it also chooses a single *special player* among all the honest ones. For all honest players which are not special, $\mathcal{S}$ runs the protocol normally. On the other hand, the role of the special player is to fix values as did P_1 in the case of static corruptions (Subsection 5.1). This explains why the security proof against adaptive corruptions very much resembles that for the static case, with some adaptations to deal with the dynamic corruption of players. As hinted previously, if the special player is corrupted, $\mathcal{S}$ rewinds the protocol; this rewind goes back to the beginning of the previous key refresh, where $\mathcal{S}$ chooses a new special player. Since there is only one special player, $\mathcal{S}$ rewinds at most $n-1$ times. If such a switching of special players (SSP) occurs, we assume that, for the duration of the Key Refresh, both the previous special player, and at least one of the remaining $n - 1$ players remain uncorrupted. Indeed while handing over the inconsistent values from the old special player to the new one, both players possess values that are inconsistent with publicly available information. However by the end of the Key Refresh in which the SSP occurs, only the new special player is inconsistent; and we can thereafter again handle $n-1$ corruptions.

We stress that since all precomputed pre-signatures are erased at every Key Refresh,[4] the aforementioned rewind does not introduce the risk that $\mathcal{A}$ may request the signature of two different messages with the same randomness.

Key Generation for $t = n - 1$. As suggested in [CGG+20], for $t = n-1$, key generation can be simplified by using an additive sharing instead of a Feldman-VSS. This improves the protocols' communication cost, speed, and simplifies the security proof. This simplified Key Generation sub-protocol is depicted in Fig. 7.

P_i	$\mathsf{IKeyGen}(\mathbb{G}, P, q)$	All players $\{P_j\}_{j\neq i}$
$w_i \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$; $W_i \leftarrow w_i P$		
$[\mathsf{kgc}_i, \mathsf{kgd}_i] \leftarrow \mathsf{Com}(W_i)$		
$(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{CL.KeyGen}(1^\lambda)$	$\xRightarrow{\mathsf{pk}_i \text{ and } \mathsf{kgc}_i}$	
	$\xRightarrow{\mathsf{kgd}_i}$	$Q = \sum_{i=1}^{n} W_i$
$\pi_{\mathsf{kg},i} := \mathsf{ZKPoK}_{W_i}\{(w_i) : W_i = w_i P\}$	$\xleftrightarrow{\pi_{\mathsf{kg},i}}$	

Fig. 7: Key Generation protocol when $t = n - 1$

On the adaptive security of Feldman-VSS. We here give an idea why attaining security against adaptive adversaries for our full threshold protocol (any $t < n$) is considerably more challenging. As mentioned above, to guarantee adaptive security, $\mathcal{S}$ must be capable of providing a consistent internal state whenever $\mathcal{A}$ corrupts an honest player. We point out that using Feldman-VSS causes issues in the simulation, since the simulator – simulating the special player P_* – computes a polynomial $p_*(X)$, for which it can give at most t consistent shares which pass the verification check. Indeed, $t+1$ shares define in a unique way $p_*(X)$, which has an unknown degree zero coefficient. As a result, $\mathcal{S}$ will send t consistent shares and $n-t$ inconsistent shares with overwhelming probability.

In the case of static corruptions this is not a problem since the consistent shares are given to the adversary, while the inconsistent ones are given to the honest players, that will not be corrupted. In contrast, if the adversary is adaptive, $\mathcal{S}$ does not know which players $\mathcal{A}$ will corrupt, so it sends the t consistent values to t random players. If the t players $\mathcal{A}$ chooses to corrupt do not coincide with the t players having received consistent values, then $\mathcal{A}$ has corrupted a player with an inconsistent internal state and hence distinguishes real and simulated executions. As explained in the *proof strategy* paragraph, each time an

[4] In fact all randomness and data used in the previous refreshment phase is erased, except for the information that the protocol specifies should be used afterwards.

honest player is corrupted revealing inconsistent values, $\mathcal{S}$ rewinds the protocol. Therefore $\mathcal{S}$'s running time (which must be polynomial for security to hold) grows exponentially with $n - t$. For $t = n - 1$ (the setting we consider), there is only one inconsistent share, that of the special player, hence the number of potential rewinds remains reasonable.

6 Efficiency comparisons

We here compare the theoretical complexity of our protocol to that of [CGG$^+$20] for the standard NIST curve P-256 corresponding to 128 security level. For the encryption scheme, we start with a 112 bit security as in [CGG$^+$20], but also study the case where its level of security matches that of the elliptic curve.

The figures we provide count the number of group and ring elements which are both sent and received from a given party, including broadcasts; whereas the figures provided in [CGG$^+$20, Fig 1] only include the data sent from one player to another. We focus on pre-signing and signing sub-protocols; these are the most critical as they will be most frequently executed.

We compute the communication costs for both protocols presented in [CGG$^+$20]; one which benefits of only having three rounds, and their six round protocol which benefits of a more efficient identification procedure if an abort occurs. Regarding our work, computations are based on the sub-protocols described in Section 3. The resulting figures are provided in Fig. 8. For our choice on the size of the discriminant Δ_K defining the class group and the resulting number of bits required to represent elements, we refer the reader to [CCL$^+$20]. We further reduce the representation of class group elements by a factor 3/4 by relying on the simple yet elegant compression technique presented by Dobson et al in [DGS20]. Fig. 8 clearly demonstrates the impressive efficiency gains we attain, reducing by a factor 10 the bandwidth consumption compared to [CGG$^+$20].

Protocol	Curve size	λ (bits)	Δ_K (bits)	N (bits)	Total Signing (KBytes)
Canetti et al.'s 6 rounds	256	112	-	2048	$31.3t + 1.0$
Canetti et al.'s 3 rounds	256	112	-	2048	$31.6t + 1.3$
Ours	256	112	1348	-	$\mathbf{3.4t + 2.0}$
Canetti et al.'s 6 rounds	256	128	-	3072	$45.1t + 1.5$
Canetti et al.'s 3 rounds	256	128	-	3072	$45.3t + 1.8$
Ours	256	128	1827	-	$\mathbf{4.1t + 2.3}$

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

Comparing Key Refresh. One of the main benefits of our protocol compared to that of Canetti *et al.* is the huge improvement provided by our Key Refresh protocol for reasonable numbers of users. Precisely, in [CGG$^+$20] each player is required to generate a new Paillier's (RSA) modulus N together with a proof that this was constructed correctly. For a 112 bits level of security, in their n out of n Key Refresh protocol requires essentially $5n^2 + n$ elements of size $|q|$ and $n^2 + 163n$ ring elements (of size $2|N|$ to be sent between all players. Whereas our Key Refresh requires $3(n^2 + n)$ elements of size $|q|$; $4n^2 + 5n$ group elements from the class group and $n^2 + n$ challenges, of size $|\Delta_K|/2 + 80 + \lambda$. Concretely, for $n = 5$, $\lambda = 112$ this results in 420KBytes of data being transmitted in their protocol, as opposed to 28KBytes in ours; i.e. a reduction by a factor 15.

References

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

BKSW20. K. Belabas, T. Kleinjung, A. Sanso, and B. Wesolowski. A note on the low order assumption in class group of an imaginary quadratic number fields. Cryptology ePrint Archive, Report 2020/1310, 2020.

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

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

CCL$^+$20. G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker. Bandwidth-efficient threshold EC-DSA. In *PKC 2020, Part II*, *LNCS* 12111, pages 266–296. Springer, Heidelberg, 2020.

CGG$^+$20. R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled. Uc non-interactive, proactive, threshold ecdsa with identifiable aborts. In *ACM CCS 2020*, CCS '20, page 1769–1787, New York, NY, USA, 2020. ACM.

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

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

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

CP93. D. Chaum and T. P. Pedersen. Wallet databases with observers. In *CRYPTO'92*, *LNCS* 740, pages 89–105. Springer, Heidelberg, 1993.

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

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

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

DGS20. S. Dobson, S. D. Galbraith, and B. Smith. Trustless groups of unknown order with hyperelliptic curves. Cryptology ePrint Archive, Report 2020/196, 2020.

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

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

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

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

GG20. R. Gennaro and S. Goldfeder. One round threshold ecdsa with identifiable abort. Cryptology ePrint Archive, Report 2020/540, 2020.

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

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

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

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

IOZ14. Y. Ishai, R. Ostrovsky, and V. Zikas. Secure multi-party computation with identifiable abort. In *CRYPTO 2014, Part II*, *LNCS* 8617, pages 369–386. Springer, Heidelberg, 2014.

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

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

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

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

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

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

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

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

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

Auxiliary Supporting Material

I Feldman's Verifiable Secret Sharing

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

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

1. generate a random degree t polynomial $p \in \mathbf{Z}/q\mathbf{Z}[x]$ with free term σ; i.e.:

$$p(x) = a_t x^t + a_{t-1}x^{t-1} + \ldots + a_2 x^2 + a_1 x + \sigma \mod q.$$

   The shares of σ are $\sigma_i = p(i) \mod q$.
2. send σ_i to P_i, for all i.
3. publishe auxiliary information allowing other players to check the shares are consistent and define a unique secret: $\{v_i = g^{a_i} \in \mathbb{G}\}_{i\in[t]}$ and $v_0 = g^{\sigma} \in \mathbb{G}$.

All parties check their share is consistent by verifying that the following holds:

$$g^{\sigma_i} = \prod_{j=0}^{t} v_j^{i^j} \in \mathbb{G}.$$

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

II Equivocable Commitment Schemes

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

Formally a (non-interactive) equivocable commitment scheme consists of four PPT algorithms: (1) $\mathsf{Setup}(1^\lambda) \to (\mathsf{pp}, \mathsf{tk})$ which outputs public parameters pp and associated secret trapdoor key tk; (2) $\mathsf{Com}(m, r) \to [\mathsf{c}(m), \mathsf{d}(m)]$ which on input a message m and random coins r, outputs the commitment $\mathsf{c}(m)$ and an opening value $\mathsf{d}(m)$ (if S refuses to open a commitment $\mathsf{d}(m) = \perp$); (3)

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

III Arguing knowledge of a decrypted message

Consider a party P with encryption key pair $(\mathsf{pk}, \mathsf{sk})$. And suppose P has received a ciphertext $\vec{c} = (c_1, c_2) \in \widehat{G}^2$, encrypted under pk. The following proof allows P to convince a verifier that decrypting $\vec{c}$ with sk yields m, which may either be $\bot$ if $\vec{c}$ fails to decrypt, or some value in $\mathbf{Z}/q\mathbf{Z}$.

Note that c may or may not be valid (i.e. $c_1 \in G^q$); this is irrelevant, indeed P himself may not know if it is the case, so there is no way P can prove it is so.

Let $M := c_2 \cdot c_1^{-\mathsf{sk}}$. Note that if $M = f^a$ for some $a \in \mathbf{Z}/q\mathbf{Z}$, the decryption algorithm returns a, otherwise it returns $\bot$. Observe that when P is proving that $\vec{c}$ decrypts to a given value, P can reveal M to all players, so everyone can compute $c_2 \cdot M^{-1}$. We present a ZKAoK for the following relation:

$$\mathsf{R}_{\mathsf{Dec}} := \{(\mathsf{pk}, (c_1, c_2), M); \mathsf{sk} \mid c_1, c_2, M \in \widehat{G};\ c_2 \cdot M^{-1} = c_1^{\mathsf{sk}} \wedge \mathsf{pk} = g_q^{\mathsf{sk}}\}.$$

The interactive protocol is given in Fig. 1. We denote $\mathcal{C}$ the challenge set, and $C := |\mathcal{C}|$. The only constraint on C is that the C-low order assumption holds in $\widehat{G}$. The protocol is complete, honest verifier zero-knowledge, and sound under the assumption the strong root problem for class groups with input $(\widehat{G}, \widehat{G}^q, g_q)$, and the C-low order problem in $\widehat{G}$ are hard. The proof is essentially that of [CCL+20, Theorem 2] for their ZKAoK for relation $\mathsf{R}_{\mathsf{Enc}}$ with very minor variations.

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 $g_q := \widehat{g}_q^t$.

$$\begin{array}{ccc}
\text{Prover } (\mathsf{pk}, (c_1, c_2), M; \mathsf{sk}) & & \text{Verifier } (\mathsf{pk}, (c_1, c_2), M) \\
\hline
r \xleftarrow{\$} [2^{40}\tilde{A}C] & & \\
t_1 := c_1^r \text{ and } t_2 := g_q^r & \xrightarrow{\quad t_1, t_2 \quad} & \\
 & & k \xleftarrow{\$} C \\
 & \xleftarrow{\quad k \quad} & \\
u := r + k \cdot \mathsf{sk} \in \mathbf{Z} & & \\
 & \xrightarrow{\quad u \quad} & \text{Check } u \in [\tilde{A}C(2^{40}+1)] \\
 & & \text{and } c_1^u = t_1 \cdot (c_2 \cdot M^{-1})^k \\
 & & \text{and } g_q^u = t_2 \cdot \mathsf{pk}^k \\
\hline
\end{array}$$

Fig. 1: Zero-knowledge argument of knowledge for $\mathsf{R_{Dec}}$.

IV Indistinguishability of real and simulated semi-correct executions (static corruptions)

Lemma 2 *Assuming the strong root assumption and the C-low order assumption hold for Gen; the CL encryption scheme is δ_s-smooth; 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.*

The proof is very similar to that in [CCL$^+$20], the main difference being that $\mathcal{S}$ now also simulates the identification procedure when the protocol aborts. Before proving Lemma 2, let us define the notion of invalid ciphertexts.

Definition 1. *A ciphertext is said to be invalid if it is of the form $(u, e) := (u, u^{\mathsf{sk}} f^m)$ where $u \in G \backslash G^q$. Note that one can compute such a ciphertext using the secret key sk, but not the public key pk; that the decryption algorithm applied to (u, e) with secret key sk recovers m; and that an invalid ciphertext is indistinguishable of a valid one under the hardness of HSM.*

Indistinguishability of identification procedure. See Section 5.2.

Indistinguishability of pre-signing and signing protocols. The differences between $\mathcal{A}$'s real and simulated view are the following:

1. $\mathcal{S}$ does not know w_1. So for each $j \in S, j \neq 1$ it cannot compute $c_{k_j w_1}$ as in a real execution of the protocol. However under the strong root and C-low order assumption in $\widehat{G}$, $\mathcal{S}$ can extract k_j from proof π_j in Phase 1 for each $j \in$

$S, j \neq 1$. $\mathcal{S}$ needs to simulate P_1 as a respondent in MtAwc protocols, then it chooses a random $\mu_{j,1}$ and encrypt it as we have seen in Phase 2 simulation. The resulting view of $\mathcal{A}$ is identical to an honestly generated one since both in real and simulated executions $\mu_{j,1}$ is uniformly distributed in $\mathbf{Z}/q\mathbf{Z}$. Moreover 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 $\mathcal{A}$'s view.

2. $\mathcal{S}$ computes $\widehat{\Gamma}_1 := \widetilde{\delta} \cdot R - \sum_{i \in S, i \neq 1} \Gamma_i$, and equivocates its commitment c_1 s.t. d_1 decommits to $\widehat{\Gamma}_1$. Let us denote $\widehat{\gamma}_1 \in \mathbf{Z}/q\mathbf{Z}$ the value s.t. $\widehat{\Gamma}_1 = \widehat{\gamma}_1 P$, where $\widehat{\gamma}_1$ is unknown to $\mathcal{S}$, but the forger can simulate the ZKPoK of $\widehat{\gamma}_1$. Let us further denote $\widehat{k} \in \mathbf{Z}/q\mathbf{Z}$ the randomness (unknown to $\mathcal{S}$) used by its' signing oracle to produce R. It holds that $\widetilde{\delta} = \widehat{k}(\widehat{\gamma}_1 + \sum_{j \in S, j \neq 1} \gamma_j)$, where $\Gamma_i = \gamma_i \cdot P$, to distinguish them from the $\widetilde{\gamma}_i$s which are used in MtA protocols. Finally, let us denote $\widehat{k}_1 := \widehat{k} - \sum_{j \in S, j \neq 1} k_j$. $\mathcal{S}$ is implicitly using $\widehat{k}_1 \neq k_1$, even though $\mathcal{A}$ received an encryption of k_1 in Phase 1. However, from the smoothness of the CL scheme, and the hardness of the HSM problem, this change is unnoticeable to $\mathcal{A}$.

Claim. If the CL encryption scheme is δ_s-smooth and the HSM problem is δ_{HSM}-hard, then no probabilistic polynomial time adversary $\mathcal{A}$ – interacting with $\mathcal{S}$ – can notice the value of k_1 in the computation of R being replaced by the (implicit) value $\widehat{k}$ with probability greater than $2\delta_{\mathsf{HSM}} + 3/q + 4\delta_s$.

Proof. To see this consider the following sequence of games. We denote E_i the probability $\mathcal{A}$ outputs 1 in $\mathsf{Game_i}$.

Game_0 *to* Game_1. $\mathcal{S}$ uses the secret key sk_1 instead of the public key pk_1 and r_1 to compute $c_{k_1} \leftarrow (u_1, u_1^{\mathsf{sk}_1} f^{k_1})$ where $u_1 = g_q^{r_1}$. The simulation of honest players uses the public key as usual. Both games are perfectly indistinguishable from $\mathcal{A}$'s view:

$$|\Pr[\mathsf{E}_1] - \Pr[\mathsf{E}_0]| = 0.$$

Game_1 *to* Game_2. In Game_2 one replaces the first element of c_{k_1} (in Game_1 this is $u_1 \in G^q$) with $\widetilde{u}_1 \in G \backslash G^q$. There exists a unique $r_1 \in \mathbf{Z}/s\mathbf{Z}$ and $b_1 \in \mathbf{Z}/q\mathbf{Z}$ such that $\widetilde{u}_1 = g_q^{r_1} f^{b_1}$. And $c_{k_1} = (\widetilde{u}_1, \widetilde{u}_1^{\mathsf{sk}_1} f^{k_1})$. Under the δ_{HSM}-hardness of HSM both games are indistinguishable:

$$|\Pr[\mathsf{E}_2] - \Pr[\mathsf{E}_1]| \leqslant \delta_{\mathsf{HSM}}.$$

Game_2 *to* Game_3. In Game_3 the points $Q = x \cdot P$ and $R = \widehat{k}^{-1} \cdot P$ come from the EC-DSA oracle, while in Game_2 they are computed as in the real protocol. As a result, the value k_1 encrypted in c_{k_1} is unrelated to $\widehat{k}$. Let us denote $\widehat{k}_1 := \widehat{k} - \sum_{j \in S, j \neq 1} k_j$, this is the value that – if used by $\mathcal{S}$ instead of k_1 – would lead to the joint computation of $R = \widehat{k}^{-1} P$.

To demonstrate that Game_2 and Game_3 are indistinguishable from $\mathcal{A}$'s view, we start by considering a fixed $\widehat{\mathsf{sk}}_1 \in \mathbf{Z}$ satisfying the following equations:

$$\begin{cases} \widehat{\mathsf{sk}}_1 \equiv \mathsf{sk}_1 \bmod \varpi, \\ \widehat{\mathsf{sk}}_1 \equiv \mathsf{sk}_1 + {b_1}^{-1}(k_1 - \widehat{k}_1) \bmod q, \end{cases}$$

where ϖ is the group exponent of $\widehat{G}$, such that the order s of g_q divides ϖ. Note that the smoothness of the CL encryption scheme ensures that such a $\widehat{\mathsf{sk}}_1$ exists (it is not necessarily unique). We can now see that in Game_3, c_{k_1} is an invalid encryption of both $\widehat{k}_1$ and of k_1, for respective secret keys $\widehat{\mathsf{sk}}_1$ and sk_1, but for the same public key pk_1, indeed:

$$\begin{aligned} c_{k_1} &= (\widetilde{u}_1, \widetilde{u}_1^{\mathsf{sk}_1} f^{k_1}) = (g_q^{r_1} f^{b_1}, (g_q^{r_1} f^{b_1})^{\mathsf{sk}_1} \cdot f^{k_1}) \\ &= (g_q^{r_1} f^{b_1}, \mathsf{pk}_1^{r_1} f^{\widehat{\mathsf{sk}}_1 \cdot b_1 + \widehat{k}_1}) = (\widetilde{u}_1, \widetilde{u}_1^{\widehat{\mathsf{sk}}_1} f^{\widehat{k}_1}). \end{aligned}$$

Adversary $\mathcal{A}$ receives the point Q, the encryption key $\mathsf{pk}_1 = g_q^{\mathsf{sk}_1}$, and c_{k_1} from $\mathcal{S}$ (at this point $\mathcal{A}$ view is identical to that in Game_2). Now $\mathcal{A}$ corrupting P_j computes $c_{k_1 \gamma_j}$ which we denote $c_\alpha = (u_\alpha, e_\alpha)$, and $c_{k_1 w_j}$ which we denote $c_\mu = (u_\mu, e_\mu)$. $\mathcal{A}$ then sends c_α and c_μ to $\mathcal{S}$. The difference between Game_2 and Game_3 appears now in how $\mathcal{S}$ attempts to decrypt c_α and c_μ. In Game_2 it would have used $\widehat{\mathsf{sk}}_1$, whereas in Game_3 it uses sk_1.

Notation. We denote α (resp. μ) the random variable obtained by decrypting c_α (resp. c_μ) (received in Game_3) with decryption key sk_1; we denote α' (resp. μ') the random variable obtained by decrypting c_α (resp. c_μ) (received in Game_3) with decryption key $\widehat{\mathsf{sk}}_1$; we introduce a hypothetical $\mathsf{Game}_3{}'$, which is exactly as Game_3, only one decrypts c_α (resp. c_μ) (received in Game_3) with decryption key $\widehat{\mathsf{sk}}_1$, thus obtaining α' (resp. μ'). Moreover in Game 3$'$ the check performed on the curve is 'If $\mu' \cdot P + B_{1,j} \neq \widehat{k}_1 \cdot W_j$ then abort'.

Observation. The view of $\mathcal{A}$ in Game_2 and in $\mathsf{Game}_3{}'$ is identical. By demonstrating that the probability $\mathcal{A}$'s view differs when $\mathcal{S}$ uses α, μ in Game_3 from when it uses α', μ' in $\mathsf{Game}_3{}'$ is negligible, we can conclude that $\mathcal{A}$ cannot distinguish Game_2 and Game_3 except with negligible probability.
The smoothness of the CL encryption scheme tells us that given pk_1, which fixes (sk_1 mod s), the value of (sk_1 mod q) remains δ-close to the uniform distribution modulo q. In particular this ensures that $\mathcal{A}$'s view of α and α' are δ-close. Indeed, $\mathcal{A}$ receives an invalid encryption of k_1, which information theoretically masks k_1. At this point $\mathcal{A}$'s view of k_1 is that of a random variable δ-close to the uniform distribution modulo q. $\mathcal{A}$ then computes c_α which it sends to $\mathcal{S}$. Finally $\mathcal{A}$ receives either (a one way function of) k_1, or (a one way function of) some random value which is unrelated to k_1, and must decide which it received. For μ and μ', the indistinguishability of $\mathcal{A}$'s view of both random variables is a little more delicate, since $\mathcal{A}$ gets additional information from the check on the curve performed by $\mathcal{S}$, namely in Game_3 if $\mu \cdot P + B_{1,j} \neq k_1 \cdot W_j$ the simulator aborts. We call the output of this check test. And in $\mathsf{Game}_3{}'$, if $\mu' \cdot P + B_{1,j} \neq \widehat{k}_1 \cdot W_j$ the simulator aborts. We call the output of this check test'. Notice that if $\mathsf{test} = \mathsf{test}'$, both games are δ_s-close from $\mathcal{A}$'s view (the only change is in the ciphertext c_{k_1}). Let us bound the probability $\mathfrak{p}$ that $\mathsf{test} \neq \mathsf{test}'$. This will allow us to conclude that

$$|\Pr[\mathsf{E}_3] - \Pr[\mathsf{E}_2]| \leq \mathfrak{p} + \delta_s.$$

Let us consider the ciphertext $c_\mu = (u_\mu, e_\mu) \in \widehat{G} \times \widehat{G}$ sent by $\mathcal{A}$. There exist unique $z_\mu \in \widehat{G}^q$, $y_\mu \in F$ such that $u_\mu = z_\mu y_\mu$. Moreover there exists a unique $b_\mu \in \mathbf{Z}/q\mathbf{Z}$ such that $y_\mu = f^{b_\mu}$.

Since $\mathsf{sk}_1 = \widehat{\mathsf{sk}}_1 \bmod \varpi$, $\mu = \perp$ if and only if $\mu' = \perp$, and this occurs when $e_\mu \cdot z_\mu^{-\mathsf{sk}_1} = e_\mu \cdot z_\mu^{-\widehat{\mathsf{sk}}_1} \notin F$. In this case Game_3 is identical to $\mathsf{Game}_3{}'$ from $\mathcal{A}$'s view ($\mathcal{S}$ aborts in both cases). We hereafter assume decryption does not fail, which allows us to adopt the following notation $e_\mu = z_\mu^{\mathsf{sk}_1} f^{h_\mu} = z_\mu^{\widehat{\mathsf{sk}}_1} f^{h_\mu}$ with $h_\mu \in \mathbf{Z}/q\mathbf{Z}$. We thus have:

$$\mu := \log_f\left(\frac{e_\mu}{u_\mu^{\mathsf{sk}_1}}\right), \quad = h_\mu - b_\mu \mathsf{sk}_1 \bmod q \qquad \text{and} \qquad \mu' := \log_f\left(\frac{e_\mu}{u_\mu^{\widehat{\mathsf{sk}}_1}}\right), \quad = h_\mu - b_\mu \widehat{\mathsf{sk}}_1 \bmod q$$

Thus we have

$$\mu - \mu' \equiv b_\mu(\widehat{\mathsf{sk}}_1 - \mathsf{sk}_1) \equiv b_\mu {b_1}^{-1}(k_1 - \widehat{k}_1) \bmod q.$$

We consider three cases:

(a) $\mu = \mu' \bmod q$. This may happen for two reasons:
   i. If $k_1 \equiv \widehat{k}_1 \bmod q$, then Game_2 and Game_3 are identical.
   ii. Else $b_\mu = 0 \bmod q$, i.e. c_μ is a valid ciphertext. Since we ruled out $k_1 \equiv \widehat{k}_1 \bmod q$ in the previous case, if test=true, necessarily test'=false, and vis versa. Both cases being symmetric, we consider the case test=true. From $\mathcal{A}$'s view, before outputting c_μ the only fixed information relative to k_1 is that contained $c_{k_1} = (g_q^{r_1} f^{b_1}, (g_q^{r_1} f^{b_1})^{\mathsf{sk}_1} f^{k_1})$. This fixes $\pi_0 := b_1 \cdot \mathsf{sk}_1 + k_1 \bmod q$. However from $\mathcal{A}$'s view, given pk_1, the random variable sk_1 follows a distribution δ_s-close to $\mathcal{U}(\mathbf{Z}/q\mathbf{Z})$. Thus k_1 also follows a distribution δ_s-close to $\mathcal{U}(\mathbf{Z}/q\mathbf{Z})$. Now suppose $\mathcal{A}$ returns $c_\mu = (z_\mu, z_\mu^{\mathsf{sk}_1} f^\mu)$ where $z_\mu \in \widehat{G}^q$. If $\mathsf{test} = \mathsf{true}$, then $\mu \cdot P + B_{1,j} = k_1 W_j$, and $\mathcal{A}$ has fixed the correct value of k_1, this occurs with probability $\leqslant 1/q + \delta_s$.

(b) $\mu \not\equiv \mu' \bmod q$ but $\mu - \mu' = w_j(k_1 - \widehat{k}_1) \bmod q$, i.e. $b_\mu = w_j b_1 \bmod q$. This results in $\mathcal{S}$ aborting on μ' in Game_2 if and only if $\mathcal{S}$ aborts on μ in Game_3. This occurs if the adversary performs homomorphic operations on c_{k_1}, and the difference between the random variables is that expected by $\mathcal{S}$. Indeed:

$$\mu = k_1 w_j - \nu_{1,j} \Leftrightarrow \mu' + w_j(k_1 - \widehat{k}_1) = k_1 w_j - \nu_{1,j} \Leftrightarrow \mu' = \widehat{k}_1 w_j - \nu_{1,j}.$$

(c) ($\mu \not\equiv \mu' \bmod q$) and ($\mu - \mu' \not\equiv w_j(k_1 - \widehat{k}_1) \bmod q$). We here consider three sub-cases:
   i. Either $\mathsf{test} = \mathsf{test}' = \mathsf{false}$; this results in identical views for $\mathcal{A}$.
   ii. Either $\mathsf{test}' = \mathsf{true}$; this means that:

$$\mu' = \widehat{k}_1 w_j - \nu_{1,j} \bmod q.$$

Now since $\mu - \mu' \neq w_j(k_1 - \widehat{k}_1) \bmod q$ necessarily $\mathsf{test} = \mathsf{false}$. Consequently if this event occurs, $\mathcal{A}$'s view differs. Let us prove that information theoretically, this can not happen with probability greater than $1/q + \delta_s$. To this end we consider the distribution followed by the point $P := (\mathsf{sk}_1, \widehat{\mathsf{sk}}_1, k_1, \widehat{k}_1) \in (\mathbf{Z}/q\mathbf{Z})^4$, conditioned on $\mathcal{A}$'s view. For clarity, we first recall the expression of c_{k_1} received by $\mathcal{A}$:

$$c_{k_1} = (g_q^{r_1} f^{b_1}, \mathsf{pk}_1^{r_1} f^{\widehat{\mathsf{sk}}_1 b_1 + \widehat{k}_1})$$

where $b_1 \neq 0 \bmod q$. We also recall the expression of c_μ, sent by $\mathcal{A}$ to $\mathcal{S}$. Since c_μ decrypts to μ' with decryption key $\widehat{\mathsf{sk}}_1$, we can write:

$$c_\mu = (z_\mu f^{b_\mu}, z_\mu^{\widehat{\mathsf{sk}}_1} f^{\mu' + b_\mu \widehat{\mathsf{sk}}_1}).$$

Let us denote $\pi_0 := \widehat{\mathsf{sk}}_1 b_1 + \widehat{k}_1 \bmod q$ and $\pi_1 := \mu' + b_\mu \widehat{\mathsf{sk}}_1$. For this case to occur, it must hold that $\mu' = \widehat{k}_1 w_j - \nu_{1,j} \bmod q$, so

$$\pi_1 = \widehat{k}_1 w_j - \nu_{1,j} + b_\mu \widehat{\mathsf{sk}}_1 \bmod q.$$

Substituting $\widehat{\mathsf{sk}}_1$ for $(\pi_0 - \widehat{k}_1) b_1^{-1}$ yields:

$$\begin{aligned} \pi_1 &= \widehat{k}_1 w_j - \nu_{1,j} + b_\mu b_1^{-1}(\pi_0 - \widehat{k}_1) \bmod q \\ \Leftrightarrow \pi_1 + \nu_{1,j} - b_\mu b_1^{-1} \pi_0 &= \widehat{k}_1 (w_j - b_\mu b_1^{-1}) \bmod q \end{aligned}$$

As we dealt with $b_\mu = w_j b_1 \bmod q$ in case (b), here $w_j - b_\mu b_1^{-1}$ is invertible mod q so we can write:

$$\widehat{k}_1 = (\pi_1 + \nu_{1,j} - b_\mu b_1^{-1} \pi_0)(w_j - b_\mu b_1^{-1})^{-1} \bmod q \qquad (1)$$

where π_0, b_1 are fixed by c_{k_1}; π_1, b_μ are fixed by c_μ; w_j is fixed by W_j; and $\nu_{1,j}$ is fixed by $B_{1,j}$. So given $\mathcal{A}$'s view and $\mathcal{A}$'s output ($B_{1,j}$ and c_μ), all the terms on the right hand side of Eq. 1 are fixed. However, given pk_1, c_{k_1} and W_j (which is all the relevant information $\mathcal{A}$ gets prior to outputting c_μ), the δ_s-smoothness of the projective hash family ensures that $\widehat{k}_1$ follows a distribution δ_s-close to $\mathcal{U}(\mathbf{Z}/q\mathbf{Z})$. If the current case occurs, Eq. 1 must hold, thus from being given a view where $\widehat{k}_1$ follows a distribution δ_s-close to $\mathcal{U}(\mathbf{Z}/q\mathbf{Z})$, $\mathcal{A}$ succeeds in fixing this random variable to be the exact value used by $\mathcal{S}$. This occurs with probability $\leqslant 1/q + \delta_s$.

iii. Else $\mathsf{test} = \mathsf{true}$; this means that $\mu = k_1 w_j - \nu_{1,j} \bmod q$. Since ($\mu - \mu' \neq w_j(k_1 - \widehat{k}_1) \bmod q$) necessarily test' fails, and $\mathcal{A}$'s view differs. Reasoning as in the previous case, but setting $\pi_0 := \mathsf{sk}_1 b_1 + k_1 \bmod q$ and $\pi_1 := \mu + b_\mu \mathsf{sk}_1$, one demonstrates that this case occurs with probability $\leqslant 1/q + \delta_s$.

Combining the above, we get that $\mathsf{test}' \neq \mathsf{test}$ if and only if we are in case (a) ii. (c) ii. or (c) iii., which occurs with probability $\leqslant 3(1/q + \delta_s)$. Thus:

$$|\Pr[\mathsf{E}_3] - \Pr[\mathsf{E}_2]| \leqslant 3/q + 4\delta_s.$$

Game_3 *to* Game_4. In Game_4, the first element u_1 of c_{k_1} is once again sampled in G^q. Both games are indistinguishable under the hardness of HSM and:

$$|\Pr[\mathsf{E}_4] - \Pr[\mathsf{E}_3]| \leq \delta_{\mathsf{HSM}}.$$

Game_4 *to* Game_5*.* In Game_5 $\mathcal{S}$ uses the public key pk_1 to encrypt k_1. The change here is exactly that between Game_0 and Game_1, both games are perfectly indistinguishable, and:

$$|\Pr[\mathsf{E}_5] - \Pr[\mathsf{E}_4]| = 0.$$

Real/Ideal executions. Putting together the above probabilities, we get that:

$$|\Pr[\mathsf{E}_5] - \Pr[\mathsf{E}_0]| \leq 2\delta_{\mathsf{HSM}} + 3/q + 4\delta,$$

which concludes the proof of the claim.

3. As a consequence of the different values k and $\widehat{k}$, there is also a difference in the values $k_1 \cdot R$ and $\widehat{k_1} \cdot R = P - \sum_{i\in S, i\neq 1} k_i \cdot R$ after rewinding in phase 4. However, they follow the same distribution, and they can be distinguished if k_1 and $\widehat{k_1}$ are distinguishable in MtAwc protocols. As we have seen in point 2. this happens with negligible probability. Furthermore, since we are in a semi-correct execution, in the real protocol P_1 runs normally the zero-knowledge proof for the consistency between $k_1 \cdot R$ and c_{k_1}. In the simulated protocol, the simulator just simulates the proof for $c_{k_i 1}$ and $\hat{k}_1 \cdot R$. In each case the two worlds are indistinguishable.
4. The same reasoning in previous item can be applied to $S_1 = \sigma_1$ and $S_1 = Q - \sum_{i\in S, i\neq 1} S_i$
5. $\mathcal{S}$ 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 $s_1 = \widehat{k}m + r(\widehat{k}x - \sum_{j\in S, j\neq 1}\sigma_j)$, and $\mathcal{S}$ 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 $R = \widehat{k}^{-1}P = (\widehat{k_1} + \sum_{j\in S, j\neq 1} k_j)^{-1} \cdot P$. Therefore the value s_1 computed by $\mathcal{S}$ is consistent with a correct share for P_1 for a valid signature (r, s), which makes Phase 7 indistinguishable from the real execution to the adversary.

V Security against adaptive corruptions – Proof of theorem 2

Theorem 2 *Assuming EC-DSA is an enhanced existentially unforgeable signature scheme under chosen message attack; the* DL *assumption holds in* $\mathbb{G}$*; the* *strong root and C-low order assumptions hold for* Gen*; the* CL *encryption scheme is* $\mathsf{ind\text{-}cpa}$*-secure; and the commitment scheme is non-malleable and equivocable, then the $(n-1,n)$-threshold EC-DSA protocol of Figures 7-5-6-4 is an enhanced existentially unforgeable threshold signature scheme against adaptive corruptions.*

Notation. Before proving the theorem, let us introduce some notations. The sets of the indices of all players and all corrupted players are denoted $\mathbf{P}$ and $\mathbf{C}$ respectively. At the beginning of the experiment, the simulator randomly chooses an honest player P_* that is henceforth referred to as the special player. The set $\mathbf{H}$ contains indices of all honest players except P_*, while $\mathbf{NC}$ contains indices of all non corrupted players including P_*. Hence $\mathbf{H} = \mathbf{P} \setminus (\mathbf{C} \cup \{P_*\})$ and $\mathbf{NC} = \mathbf{P} \setminus \mathbf{C}$. The sets $\mathbf{H}, \mathbf{C}, \mathbf{NC}$ are dynamically updated with new corruptions throughout the protocol. In particular, $\mathbf{C}$ grows in size with the condition that $|\mathbf{C}| \leqslant n-1$, taking elements from $\mathbf{NC}$. Note that if P_* is corrupted, the simulator will rewind the protocol and choose a different special player. Clearly if some P is corrupted and $\mathbf{NC} \leftarrow \mathbf{NC} \setminus \{P\}$ then $\mathbf{H} \leftarrow \mathbf{H} \setminus \{P\}$. Finally, all values belonging to the special player P_* are indexed with the symbol $*$.

Simulating Key Generation.

1. $\mathcal{S}$ receives a public key Q from it's EC-DSA challenger.
2. For $i \in \mathbf{NC}$, $\mathcal{S}$ samples $w_i \overset{\$}{\leftarrow} \mathbf{Z}/q\mathbf{Z}$ and computes $[\mathsf{kgc}_i, \mathsf{kgd}_i] \leftarrow \mathsf{Com}(w_i P)$.
3. For $i \in \mathbf{NC}$, $\mathcal{S}$ samples CL encryption key pairs $(\mathsf{pk}_i, \mathsf{sk}_i) \overset{\$}{\leftarrow} \mathsf{KeyGen}(1^\lambda)$.
4. $\mathcal{S}$ broadcasts $\{\mathsf{kgc}_i\}_{i\in\mathbf{NC}}$ and $\{\mathsf{pk}_i\}_{i\in\mathbf{NC}}$, before receiving $\{\mathsf{kgc}_j\}_{j\in\mathbf{C}}$ and the public keys $\{\mathsf{pk}_j\}_{j\in\mathbf{C}}$ from $\mathcal{A}$.
5. $\mathcal{S}$ broadcasts $\{\mathsf{kgd}_i\}_{i\in\mathbf{NC}}$ and receives $\{\mathsf{kgd}_j\}_{j\in\mathbf{C}}$. For $i \in \mathbf{P}$, let $W_i \leftarrow \mathsf{Open}(\mathsf{kgc}_i, \mathsf{kgd}_i)$ be the revealed commitment value of each party.
6. $\mathcal{S}$ chooses a special player P_* and rewinds $\mathcal{A}$ to the decommitment step, so as to equivocate P_*'s commitment to $\widehat{\mathsf{kgd}}_*$ which decommits to $\widehat{W}_* := Q - \sum_{j\neq *} W_j$.
7. $\mathcal{S}$ simulates π_*^{kg} (the ZKPoK that it knows w_* corresponding to $\widehat{W}_*$) and honestly performs the proofs π_i^{kg} for $i \in \mathbf{H}$. Then, for $j \in \mathbf{C}$, $\mathcal{S}$ receives from $\mathcal{A}$ a ZKPoK of w_j satisfying $W_j := w_j \cdot P$; from which $\mathcal{S}$ can extract w_j.

Simulating Key Refresh. In the event of a normal Key Refresh (i.e. which is not due to $\mathcal{S}$ rewinding to switch special players), $\mathcal{S}$ simply runs the real Key Refresh sub-protocol for all players in $\mathbf{NC}$.
Switching Special Players in Key Refresh. As explained in the paragraph entitled *Proof Strategy* of Subsection 5.3, if at any point during the simulation $\mathcal{A}$ corrupts the special player P_*, then $\mathcal{S}$ rewinds the adversary and chooses a new special player P_*^{new} among the honest parties P_i for $i \in \mathbf{H}$. We will hereafter refer to this particular simulation of the Key Refresh protocol as *Key Refresh with special*

player switch (KRSS). At the end of the KRSS, the previous special player P_* has consistent values, so that P_*^{new} is the unique inconsistent (i.e. special) player. Note that throughout KRSS, we assume P_* is not corrupted, and $\mathcal{A}$ can corrupt at most $n-2$ of the remaining $n-1$ players.

Without loss of generality we set $P_1 := P_*$ and $P_2 := P_*^{\mathsf{new}}$. For $i \in \mathbf{P}$ we denote $u_i \in \mathbf{Z}/q\mathbf{Z}$ the secret share of the EC-DSA signing key x owned by P_i from the previous Key Refresh; and $Q_i := u_i P$. Recall that $\mathcal{S}$ does not know u_1. If a KRSS occurs, $\mathcal{S}$ simulates P_1 and P_2 in the following way (the simulation remains the same for other players):

- Sample $v_{1,1}, \ldots, v_{1,n}$ and $v_{2,1}, \ldots, v_{2,n}$ as per the protocol.
- Sample a random $\alpha \overset{\$}{\leftarrow} \mathbf{Z}/q\mathbf{Z}$ and let $\beta := v_{1,1} + v_{1,2} - \alpha$.
  Then set $Q_{1,1} := -Q_1 + \alpha P$, $Q_{1,2} := Q_1 + \beta \cdot P$ and for each $j \in \mathbf{P}, j > 2$ set $Q_{1,j} := v_{1j} \cdot P$.
- Compute $Q_{2,j} := v_{2,j} \cdot P$ for all $j \in \mathbf{P}, j \neq 2$ as in the real protocol.
- For $j \in \mathbf{P}$ compute ciphertexts $C_{1,j} \leftarrow \mathsf{Enc}(\mathsf{pk}_j, v_{1,j})$ and $C_{2,j} \leftarrow \mathsf{Enc}(\mathsf{pk}_j, v_{2,j})$ as per the protocol. Simulate proofs $\pi^{\mathsf{kr}}_{1,1}$ and $\pi^{\mathsf{kr}}_{1,2}$, but run all other proofs $\{\pi^{\mathsf{kr}}_{1,j}\}_{j \in \mathbf{P}, j>2}$ and $\{\pi^{\mathsf{kr}}_{2,j}\}_{j \in \mathbf{P}}$ as in the real protocol.
- After having received all the $\{Q_{i,j}\}_{i \in \mathbf{P}, i>2, j \in \mathbf{P}}$, $\mathcal{S}$ computes $Q_i^{\mathsf{new}} = Q_i + \sum_{j \in \mathbf{P}} Q_{j,i}$ for each $i \neq 2$. It then rewinds the sub-protocol and changes $Q_{2,2}$ to
$$Q_{2,2} := Q - \sum_{j \neq 2} Q_1^{\mathsf{new}} - Q_2 - \sum_{i \neq 2} Q_{i,2}^{\mathsf{new}}$$
  With this choice of $Q_{2,2}^{\mathsf{new}}$, $Q_2^{\mathsf{new}} = Q_2 + \sum_i Q_{i,2}^{\mathsf{new}}$ is such that $Q = \sum_{i \in \mathbf{P}} Q_i^{\mathsf{new}}$.
- Erase all values $v_{i,j}$ and $Q_{i,j}$

Notice that with this choice of Q_1^{new}, there are no inconsistencies for P_1 and it knows the discrete log of its' point. Furthermore, thanks to the values α and β the elliptic curve points computed in an unusual way are distributed as in a real execution of the protocol.

Simulating protocols Pre-Sign and Sign. After the key generation is over, the simulator must handle the signature queries issued by $\mathcal{A}$. Recall that $\mathcal{A}$ can issue two types of queries:

- oracle $\mathcal{O}^R$ to obtain a uniformly random point $R = (r_x, r_y)$ in $\mathbb{G}$:
- oracle $\mathcal{O}^{\mathsf{Sign}(\mathsf{sk}, m; R)}$ which on input a message m chosen by $\mathcal{A}$, returns a valid signature (r, s) for m where $r := r_x \bmod q$ if $R = (r_x, r_y)$ was queried to $\mathcal{O}^R$; else it returns $\perp$.

The simulator simulates P_i for each $i \in \mathbf{NC}$ in the threshold signature protocol on input R for the offline phase (Phases 1-6), and a correct signature (r, s) for m under the public key Q for the online phase 7. We stress that though the simulator knows the decryption key sk_*, and P_*'s EC-DSA public key share $W_* = w_* \cdot P$; it does not know w_*. However the simulator knows the shares w_i of all other players ($i \in \mathbf{P} \setminus \{*\}$) from the Schnorr proofs in Key Generation phase (for $i \in \mathbf{C}$) or because it computed them (for $i \in \mathbf{H}$).

The simulation of the Pre-Signing and Signing protocols is based on [CGG$^+$20] and [CCL$^+$20], with adaptations considering previously defined dynamic sets of players ($\mathbf{NC}, \mathbf{H}, \mathbf{C}$). For each execution all parties in $\mathbf{P}$ participate. This implies that $\{w_i\}_{i\in[n]}$ are long term secrets. In the following simulation $\mathcal{S}$ aborts whenever the protocol is supposed to abort, i.e., whenever $\mathcal{A}$ refuses to decommit a committed value, a ZK proof fails, a check does not pass or if the signature (r, s) does not verify.

Simulating Pre-signing and Signing - Description of $\mathcal{S}$: For all $i \in \mathbf{H}$, i.e. honest – but not special – players P_i, $\mathcal{S}$ just runs the protocol as would P_i in a real execution. Hence in the following phases we only describe how $\mathcal{S}$ simulates P_*.

Phase 1: $\mathcal{S}$ samples $k_*, \gamma_* \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$, $r_* \xleftarrow{\$} [\widetilde{A}]$ uniformly at random. It computes $c_{k_*} \leftarrow \mathsf{Enc}(\mathsf{pk}_*, k_*; r_*)$, the associated ZKAoK π_*, and $[\mathsf{c}_*, \mathsf{d}_*] \leftarrow \mathsf{Com}(\gamma_* P)$. It broadcasts $\mathsf{c}_*, c_{k_*}, \pi_*$ before receiving $\{\mathsf{c}_j, c_{k_j}, \pi_j\}_{j\in\mathbf{C}}$ from $\mathcal{A}$. $\mathcal{S}$ checks the proofs are valid and extracts the encrypted values $\{k_j\}_{j\in\mathbf{C}}$ from which it computes $k := \sum_{i\in\mathbf{P}} k_i$.

Phase 2: Recall that during the regular run of the protocol, P_* will engage in two MtA protocols and two MtAwc protocols with each other player $P_j, j \in \mathbf{P} \setminus \{*\}$ (the corrupted players and other honest players in $\mathbf{P}$). $\mathcal{S}$ runs the protocol for P_* as follows:

(a) *Initiator for MtA with $k_i, i \in \mathbf{NC}$ and $\gamma_j, j \in \mathbf{P} \setminus \{i\}$*: $\mathcal{S}$ runs the real sub-protocol, as it knows k_i. For $j \in \mathbf{P} \setminus \{*\}$, $\mathcal{S}$ decrypts the ciphertext received from P_j obtaining $\alpha_{*,j} \mod q$ (for $j \in \mathbf{NC} \subset \mathbf{P}$ it already knows the values, however P_j may be corrupted in this phase, so $\mathcal{S}$ runs the real protocol, even between non corrupted parties).

(b) *Respondent for MtA with $k_j, j \in \mathbf{P} \setminus \{*\}$ and γ_**: $\mathcal{S}$ runs the real sub-protocol, as it knows γ_*.
Recall that $\mathcal{S}$ knows k_j from extraction in Phase 1, it also knows its own shares $\beta_{j,i}$ for $i \in \mathbf{NC}$, hence $\mathcal{S}$ can compute P_j's shares $\alpha_{j,i} = k_j\gamma_i - \beta_{j,i} \mod q$.

(c) *Initiator for MtAwc with k_* and $w_j, j \in \mathbf{P} \setminus \{*\}$*: $\mathcal{S}$ runs the real sub-protocol, as it knows k_i for $i \in \mathbf{NC}$. Notice that $\mathcal{S}$ chose w_i for $i \in \mathbf{H}$ as in the real protocol, while for $j \in \mathbf{C}$, $\mathcal{S}$ extracted w_j from π_j^{kg} in KeyGen. The only unknown share of x is the special player's w_*. For $j \in \mathbf{P}\setminus\{*\}$, $\mathcal{S}$ runs the real sub-protocol; decrypting $c_{k_* w_j}$ to obtain $\mu_{*,j}$; and checking that $\mu_{*,j}P + B_{*,j} = k_* W_j$. If so, since $\mathcal{S}$ also knows k_* and w_j, it computes $\nu_{*,j} = k_* w_j - \mu_{*,j} \mod q$.

(d) *Respondent for MtAwc with $k_j, j \in \mathbf{P}\setminus\{*\}$ and w_**: $\mathcal{S}$ knows $W_* = w_* \cdot P$ but not w_*, so it samples a random $\mu_{j,*} \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ and sets $c_{k_j w_*} \leftarrow \mathsf{Enc}(\mathsf{pk}_j, \mu_{j,*})$, and $B_{j,*} := k_j \cdot W_* - \mu_{j,*} \cdot P$. Finally $\mathcal{S}$ sends the cipertexts and the point.

Note that at this point $\mathcal{S}$ knows:

- k_i for each $i \in \mathbf{P}$, w_j for each $j \in \mathbf{P} \setminus \{*\}$
- $\alpha_{i,j}, i \in \mathbf{NC}, j \in \mathbf{P}\setminus\{i\}$ as initiator for MtA, $\alpha_{j,i}, \beta_{j,i}, i \in \mathbf{NC}, j \in \mathbf{P}\setminus\{i\}$ as respondent for MtA
- $\mu_{i,j}, \nu_{i,j}, i \in \mathbf{NC}, j \in \mathbf{P}\setminus\{i\}$ as initiator for MtAwc, $\mu_{j,i}, \nu_{j,i}, i \in \mathbf{NC}, j \in \mathbf{P} \setminus \{i\}$ as respondent for MtAwc

$\mathcal{S}$ computes δ_* for the special player and δ_i for $i \in \mathbf{H}$ as per protocol.
Note that $\mathcal{S}$ does not know the internal values from the MtA and MtAwc protocols executed by two players that are both controlled by the adversary. Thus $\mathcal{S}$ is not able to compute the individual values σ_j and δ_j for $j \in \mathbf{C}$; nor can $\mathcal{S}$ compute σ_* since it doesn't know the value w_*. However $\mathcal{S}$ can compute:

$$\begin{aligned}
\sigma_{\mathbf{C}} = \sum_{i\in\mathbf{C}} \sigma_i &= \sum_{i\in\mathbf{C}} (k_i w_i + \sum_{j\in\mathbf{P}\setminus\{i\}} \mu_{i,j} + \sum_{j\in\mathbf{P}\setminus\{i\}} \nu_{j,i}) \\
&= \sum_{i\in\mathbf{C}} \sum_{j\in\mathbf{P}\setminus\{i\}} (\mu_{i,j} + \nu_{j,i}) + \sum_{i\in\mathbf{C}} k_i w_i \\
&= \sum_{i\in\mathbf{C}} \sum_{j\in\mathbf{C}, j\neq i} (\mu_{i,j} + \nu_{j,i}) + \sum_{i\in\mathbf{C}} \sum_{j\in\mathbf{NC}} (\mu_{i,j} + \nu_{j,i}) + \sum_{i\in\mathbf{C}} k_i w_i \\
&= \sum_{i\in\mathbf{C}} \sum_{j\in\mathbf{C}, j\neq i} (\mu_{i,j} + \nu_{i,j}) + \sum_{i\in\mathbf{C}} \sum_{j\in\mathbf{NC}} (\mu_{i,j} + \nu_{j,i}) + \sum_{i\in\mathbf{C}} k_i w_i \\
&= \sum_{i\in\mathbf{C}} \sum_{j\in\mathbf{C}, j\neq i} k_i w_j + \sum_{i\in\mathbf{C}} \sum_{j\in\mathbf{NC}} (\mu_{i,j} + \nu_{j,i}) + \sum_{i\in\mathbf{C}} k_i w_i \\
&= \sum_{i\in\mathbf{C}} \sum_{j\in\mathbf{NC}} (\mu_{i,j} + \nu_{j,i}) + \sum_{i\in\mathbf{C}} \sum_{j\in\mathbf{C}} k_i w_j
\end{aligned}$$

since it knows all the values $\{k_j\}_{j\in\mathbf{P}}$, $\{w_j\}_{j\in\mathbf{P}, j\neq *}$, $\mu_{i,j}$ and $\nu_{j,i}$ in MtAwc with the honest players and $\mu_{i,*}$, $\nu_{*,i}$ from special player.
Furthermore, up until the moment $\sigma_{\mathbf{C}}$ is used to check whether the execution is semi-correct or not, every time a player P_i for some $i \in \mathbf{NC}$ is corrupted, $\mathcal{S}$ updates $\sigma_{\mathbf{C}} \leftarrow \sigma_{\mathbf{C}} + \sigma_i$. If the special player is corrupted, the simulator rewinds, and $\sigma_{\mathbf{C}}$ is recomputed.

Phase 3: $\mathcal{S}$ broadcasts δ_* and receives $\{\delta_j\}_{j\in\mathbf{C}}$ from $\mathcal{A}$. Let $\widetilde{\delta} := \sum_{i\in\mathbf{P}} \delta_i$. $\mathcal{S}$ broadcasts $T_* = \sigma_* \cdot P + \ell_* \cdot H$ ($\mathcal{S}$ can compute T_* since it knows $\sigma_* \cdot P$). As $\mathcal{S}$ does not know σ_*, it simulates the ZK proof $\tilde{\pi}_*$. Next, $\mathcal{S}$ extracts values $\widehat{\sigma}_j, \widehat{\ell}_j$ for $j \in \mathbf{C}$ from the proofs $\tilde{\pi}_j$ received from $\mathcal{A}$. Let $\widehat{\sigma}_{\mathbf{C}} := \sum_{j\in\mathbf{C}} \widehat{\sigma}_j$. Here again $\widehat{\sigma}_{\mathbf{C}}$ is updated to include $\widehat{\sigma}_i$ if a player P_i is adaptively corrupted for $i \in \mathbf{H}$.

Phase 4: $\mathcal{S}$ broadcasts d_* which decommits to Γ_*, and for all $j \in \mathbf{C}$, $\mathcal{A}$ reveals d_j which decommits to Γ_j. $\mathcal{S}$ honestly performs the ZK proof π_*^γ; and receives π_j^γ, from which $\mathcal{S}$ can extract γ_j. These are consistent with the values used in Phase 1 thanks to the binding property of the commitment scheme. Now

$\mathcal{S}$ can compute

$$\delta = (\sum_{i \in \mathbf{P}} k_i) \cdot (\sum_{i \in \mathbf{P}} \gamma_i) = k\gamma, \quad \text{where} \quad \gamma = \sum_{i \in \mathbf{P}} \gamma_i \cdot P.$$

Note that $\mathcal{A}$ may have used different values $\widetilde{\gamma}_j$ in the MtA protocol than the γ_j extracted here, hence we denote them with a tilde. At this point $\mathcal{S}$ can detect if the values published so far by $\mathcal{A}$ are consistent (the sum of the γ_j, not each individual γ_j); note that $\mathcal{S}$ will behave differently in Phases 5, 6 and 7 depending on this detection. To detect inconsistencies, $\mathcal{S}$ first computes

$$\widetilde{R} = \widetilde{\delta}^{-1} \cdot \sum_{i \in \mathbf{P}} \mathit{\Gamma}_i.$$

Then using the values $\{k_j\}_{j \in \mathbf{C}}$ extracted in Phase 1, and its own values $\{k_i\}_{i \in \mathbf{NC}}$, $\mathcal{S}$ checks if $\sum_{i \in \mathbf{P}} k_i \cdot \widetilde{R} = P$. If equality holds then $\widetilde{R} = k^{-1} \cdot P$ and $\widetilde{\delta} = k\gamma = \delta$.
The simulator can also detect if the values σ_j computed in Phase 2 are consistent with those used to compute points T_j in Phase 3; in particular $\mathcal{S}$ checks that $\widehat{\sigma}_{\mathbf{C}} = \sigma_{\mathbf{C}}$. We thus distinguish two types of executions: an execution is said to be *semi-correct* if

$$\sum_{i \in \mathbf{P}} k_i \widetilde{R} = P \qquad \text{and} \qquad \widehat{\sigma}_{\mathbf{C}} = \sigma_{\mathbf{C}}.$$

Conversely, if either of the above equalities do not hold, the execution is said to be *non semi-correct*.
Note that using EC points to check the consistency of δ and $\widetilde{\delta}$ avoids the need for proofs of affine transformation which were necessary in [CGG$^+$20] to attain security against malicious adversaries.
Now $\mathcal{S}$ adapts its behaviour depending on the type of execution:

– **Semi-correct execution:**
  (a) $\mathcal{S}$ invokes oracle $\mathcal{O}^R$ to obtain $R = (r_x, r_y)$.
  (b) $\mathcal{S}$ sets $\widehat{\mathit{\Gamma}}_* := \widetilde{\delta} \cdot R - \sum_{i \in \mathbf{P}, i \neq *} \mathit{\Gamma}_i$, so that $R = \widetilde{\delta}^{-1} \left(\widehat{\mathit{\Gamma}}_* + \sum_{i \in \mathbf{P}, i \neq *} \mathit{\Gamma}_i \right)$. Then $\mathcal{S}$ rewinds $\mathcal{A}$ to the decommitment step in Phase 4, and equivocates P_*'s commitment so that it decommits to $\widehat{\mathit{\Gamma}}_*$ instead of $\mathit{\Gamma}_*$.

– **Non semi-correct execution:** $\mathcal{S}$ simply moves on to Phase 5.

Phase 5: – **Semi-correct execution:** $\mathcal{S}$ publishes $\bar{R}_* = P - \sum_{i \in \mathbf{P} \setminus \{*\}} k_i \cdot R$ together with π'_*: a simulated ZKP of consistency with $c_{k_*} = \mathsf{Enc}(\mathsf{pk}_*, k_*; r_*)$ (note that in this case $\bar{R}_* \neq k_* \cdot R$ due to the rewinding).

– **Non semi-correct execution:** $\mathcal{S}$ publishes $\bar{R}_* := k_* \cdot R$ together with π'_*: a real ZKP of consistency with c_{k_*} (this needn't be simulated).

Phase 6: – **Semi-correct execution:** $\mathcal{S}$ publishes $S_* := Q - \sum_{j \in \mathbf{P} \setminus \{*\}} \sigma_j R$ together with π''_*: a simulated ZKP of consistency with T_* (again in this case the simulated $S_* \neq \sigma_* \cdot R$ due to the rewinding).

– **Non semi-correct execution:** $\mathcal{S}$ publishes $S_* := \sigma_* R$ together with π''_*: a real ZKP of consistency with T_* (this needn't be simulated).

In a non semi-correct execution, at least one of the the adversary's proofs π'_j or π''_j for some $j \neq *$ will fail, and the protocol will abort.

Phase 7: $\mathcal{S}$ invokes the second oracle $\mathcal{O}^{\mathsf{Sign}(\mathsf{sk},m;R)}$ with input m and R, where R was computed in one of the previous offline phases (in particular in one that was semi-correct, since it concluded successfully). In return, $\mathcal{S}$ receives the valid signature (r, s) on m, where $r = r_x \bmod q$.

At this point $\mathcal{S}$ knows $s_{\mathbf{C}} = \sum_{j\in\mathbf{C}} s_j$ (i.e., the summed value of all the s_j held by the corrupted players) because $s_{\mathbf{C}} = k_{\mathbf{C}} m + \sigma_{\mathbf{C}} r$ where $\sigma_{\mathbf{C}}$ is as defined in the simulation of Phase 2 and $k_{\mathbf{C}} = \sum_{j\in\mathbf{C}} k_j$. As in the static case, if $\mathcal{A}$ cheats in Phase 7 – denoting $\{\tilde{s}_i\}_{i\in\mathbf{C}}$ the values that $\mathcal{S}$ receives from $\mathcal{A}$ in Phase 7, and $\tilde{s}_{\mathbf{C}} := \sum_{i\in\mathbf{C}} \tilde{s}_i$ – it is possible that $s_{\mathbf{C}} \neq \tilde{s}_{\mathbf{C}}$. $\mathcal{S}$ also knows $s_{\mathbf{H}} = \sum_{i\in\mathbf{H}} s_i$ since it honestly ran the protocol for $i \in \mathbf{H}$. So $\mathcal{S}$ computes the share s_* consistent with (r, s) and $s_{\mathbf{H}\cup\mathbf{C}}$ as $s_* := s - s_{\mathbf{H}\cup\mathbf{C}}$. Finally, $\mathcal{S}$ broadcasts this value s_*.

Note on the dynamic sets. Since the set of honest and corrupted players may change throughout the protocol, if $\mathcal{S}$ has computed σ_i as an honest P_i, and P_i is subsequently corrupted, one can simply consider $i \in \mathbf{C}$, instead of $i \in \mathbf{H}$ and nothing changes. This is because once P_i is corrupted, it will be considered as malicious, with the difference that its σ_i was computed by $\mathcal{S}$ as opposed to being extracted. The proofs and checks of Phases 4, 5, and 6 ensure that σ_i does not change before Phase 7.

Simulating Identification

Simulating Identification of aborts in Key Generation – Description of $\mathcal{S}$. If an abort occurs in the Key Generation protocol, $\mathcal{S}$ runs the identification protocol as would an honest P_i for each $i \in \mathbf{NC}$ (i.e. as described on page 15. Furthermore, if some player P raises a compliant against P_* (simulated by $\mathcal{S}$), then P is detected as a cheater since the simulation key generation is done in such a way that corrupted players receive values which pass the verification check.

Simulating Identification of aborts in Pre-Sign and Sign – Description of $\mathcal{S}$. For all $i \in \mathbf{H}$, i.e. honest – but not special – players P_i, $\mathcal{S}$ just runs the identification procedure as would P_i in a real execution. Hence in the following phases we only describe how $\mathcal{S}$ simulates P_*. Consider the problematic types of abort listed on page 16. For an abort of type 1, 3 or 5 occurs, $\mathcal{S}$ runs the real identification procedure.

If an abort of type 2 occurs due to $\mathcal{S}$ announcing that the check on $\mu_{*,j}$ fails (for some $j \in \mathbf{C}$), it runs the real identification procedure. Conversely, if some player P_j for $j \in \mathbf{C}$ complains about the $\mu_{j,*}$ it received, observe that: if $\mu_{j,*}$

is the real decryption of $c_{k_j w_*}$ (which it must be if the proof for $\mathsf{R}_{\mathsf{Dec}}$ provided by P_j is valid), then since the point $B_{j,*}$ sent by $\mathcal{S}$ to P_j was computed as $B_{j,*} := k_j \cdot W_* - \mu_{j,*}$, necessarily the equality test will pass. Observe that the value $\nu_{j,*}$ remain secret in this identification protocol; hence no other (corrupted) party can check that $\mathcal{S}$ knows $\nu_{j,*}$ such that $B_{j,*} = \nu_{j,*} \cdot P$, and the simulation remains undetected.

If an abort of type 4 occurs, $\mathcal{S}$ follows the real procedure for aborts up until it needs to prove knowledge of σ_* such that $S_* = \sigma_* \cdot R$. Since $\mathcal{S}$ does not know σ_*, it simulates the proof π^*_{log}.

Indistinguishability of real and simulated executions against adaptive adversaries

The simulation of a semi-correct execution

Lemma A-1. *Assuming the strong root and C-low order assumptions hold for Gen; the CL encryption scheme is δ_s-smooth; 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 real semi-correct execution.*

The proof of Lemma A-1 very much resembles that of Lemma 2. Hence many details are here omitted.

Proof. Since, in all considered protocols, $\mathcal{S}$ simulates parties P_i for $i \in \mathbf{H}$ by running the real protocol exactly as would P_i one only needs to prove that $\mathcal{S}$'s simulation of P_* is indistinguishable from a real execution.

Indistinguishability of identification procedure in semi-correct executions. This follows immediately from the static case; it suffices to replace P_1 with P_* in the relevant paragraph in proof of Lemma 2, page 34.

Indistinguishability of Key Generation and Key refresh. For Key Generation, indistinguishability follows immediately from the static case (replacing 1 with *). Regarding Key Refresh, as long as there is no switching of special player, the simulator runs the real protocol, and the simulation is perfect. Conversely, in a KRSS, all players which are not the old or new special player are consistent; as long as neither of these is corrupted during KRSS, the simulation is perfect. If the newly chosen special player is corrupted, $\mathcal{S}$ rewinds again. And it is assumed that during a KRSS, the old special player is not corrupted.

Indistinguishability of signature protocol in semi-correct executions. The differences between $\mathcal{A}$'s real and simulated views are the following:

1. $\mathcal{S}$ does not know w_* so it cannot compute $\{c_{k_j w_*}\}_{j \in \mathbf{P} \setminus \{*\}}$ as in a real execution of the protocol. However as in the static case (replacing '1' with '*'), $\mathcal{S}$ can extract k_j from π_j for each $j \in \mathbf{C}$ and it knows k_j for each $j \in \mathbf{H}$. It then

computes the problematic ciphertexts as in the static case (*cf.* Lemma 2), and – as argued there – $\mathcal{A}$'s real and simulated view of these ciphertexts follow identical distributions.

2. $\mathcal{S}$ computes $\widehat{\Gamma}_* := \widetilde{\delta} \cdot R - \sum_{i \neq *} \Gamma_i$, and equivocates its commitment c_* s.t. d_* decommits to $\widehat{\Gamma}_*$. Once again, the proof that this change is not noticeable to $\mathcal{A}$ is identical to the static case (replacing '1' with '*', and the set S with all players $\mathbf{P}$). And using the same reasoning as in proof of Lemma 2 in Appx.IV, one can demonstrate that the following claim holds:

   Claim. If the CL encryption scheme is δ_s-smooth and the HSM problem is δ_{HSM}-hard, then no probabilistic polynomial time adversary $\mathcal{A}$ – interacting with $\mathcal{S}$ – can notice the value of k_* in the computation of R being replaced by the (implicit) value $\widehat{k}$ with probability greater than $2\delta_{\mathsf{HSM}} + 3/q + 4\delta_s$.

   Hence from the smoothness of the CL scheme, and the hardness of the HSM problem, this change is unnoticeable to $\mathcal{A}$.
3. Let us denote $\widehat{k} \in \mathbf{Z}/q\mathbf{Z}$ the randomness (unknown to $\mathcal{S}$) used by oracle $\mathcal{O}^R$ to produce R. With overwhelming probability, $k \neq \widehat{k}$. Hence there is also a difference in the values $k_* \cdot R$ and $\widehat{k}_* \cdot R = P - \sum_{i \neq *} k_i \cdot R$ after the rewind in phase 4. For the same reasons as discussed in proof of Lemma 2, item 3. (*i.e.* smoothness of encryption scheme and simulatability of the ZKP π_*, this change is indistinguishable to $\mathcal{A}$.
4. The same reasoning as in the previous item can be applied to $S_* = \sigma_* \cdot R$ and $S_* = Q - \sum_{i \neq *} S_i$.
5. $\mathcal{S}$ does not know σ_*, and thus cannot compute s_* as in a real execution. However, as in the static case, since we are in a semi-correct execution the value s_* computed by $\mathcal{S}$ is consistent with a correct share for P_* for a valid signature (r, s), which makes the simulation of Phase 7 indistinguishable from a real execution from $\mathcal{A}$'s view.

Non semi-correct executions

Once again, the proof of Lemma A-2 is essentially identical to that in the static case, substituting P_1 for P_*.

Lemma A-2. *Assuming the strong root and C-low order assumptions hold for Gen; it holds that the view of the simulation, from an adaptive adversary's view, is computationally indistinguishable from a non-semi-correct real execution.*

Combining Lemmas A-1 and A-2, it holds that if the strong root an C-low order assumptions hold for Gen; the CL encryption scheme is $\mathsf{ind\text{-}cpa}$-secure and the commitment scheme is non malleable and equivocable, then the $(n-1, n)$ EC-DSA protocol described in Figures 7,4, 5 and 6 is enhanced threshold existentially unforgeable against adaptive adversaries.