Bandwidth-efficient threshold EC-DSA revisited: Online/Offline Extensions, Identifiable Aborts Proactive and Adaptive Security
Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, Ida Tucker

To cite this version:

HAL Id: hal-03927198
https://hal.science/hal-03927198
Submitted on 6 Jan 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Bandwidth-efficient threshold EC-DSA revisited:
Online/Offline Extensions, Identifiable Aborts
Proactive and Adaptive Security

Guilhem Castagnos¹, Dario Catalano², Fabien Laguillaumie³,
Federico Savasta², and Ida Tucker⁴

¹ Université de Bordeaux, CNRS, INRIA, IMB, UMR 5251, F-33400 Talence, France
² Università di Catania, Italy.
³ LIRMM, Univ Montpellier, CNRS, Montpellier, France
⁴ IMDEA Software Institute, Madrid, Spain.

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 achieves 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 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.

Keywords: Threshold Cryptography, EC-DSA, Digital Signatures, Multi-Party Computation, Provable Security, Class Groups

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 $t = n − 1$. The protocol can also easily be modified to achieve proactive security.

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
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 $k \in \mathbb{Z}/q\mathbb{Z}$) the point $R = k^{-1}P$. Next, letting $r = r_x \mod q$ - where $R = (r_x, r_y)$ - one sets $s = kH(m) + krx \mod q$. 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 $a_ib_j$. 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 $\mathbb{Z}/q\mathbb{Z}$ 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] 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 $s_i = k_iH(m) + r(ikx)_i$. 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 over-

---

6 This result currently appears as part of the [CGG+20] paper, but it was originally made public as a separate result.
head, 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 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. 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 $(h_i = g^{sk})$ 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.

---

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 Preliminaries

Notations. For a distribution $D$, we write $d \leftarrow D$ to refer to $d$ being sampled from $D$ and $b \leftarrow B$ 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 $IP(x_1; \ldots; x_n) \rightarrow (y_1; \ldots; y_n)$ 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 $IP(x_1; \ldots; x_n) \rightarrow (y)$. In the figures representing our protocols, double arrows ($\Rightarrow$) depict a broadcast; a single arrow $P_i \rightarrow P_j$ refers to $P_i$ sending a private message to $P_j$. A (P)PT algo stands for an algorithm running in (probabilistic) polynomial time w.r.t. the length of its inputs.

2.1 Tools

Zero-knowledge proofs. A zero-knowledge proof of knowledge (ZKPoK) system for a binary relation $R$ is an interactive protocol $(P, V)$ between two probabilistic algorithms: a prover $P$ and a PT verifier $V$. Informally, $P$, detaining a witness $w$ for a given statement $x$ s.t. $(x, w) \in R$, 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:

$$ZKPoK_x\{(w) : (x, w) \in R\} \quad \text{and} \quad ZKAoK_x\{(w) : (x, w) \in R\}.$$ 

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 $P_1, \ldots, P_n$ 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 $G$ be a group of order $q$, $g$ a generator of $G$, and suppose that one of the players, that we call $P$, wants to share a secret $\sigma \in \mathbb{Z}/q\mathbb{Z}$ with the other ones. To share the secret, $P$ does the following steps:

1. generate a random degree $t$ polynomial $p \in \mathbb{Z}/q\mathbb{Z}[x]$ with free term $\sigma$; 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 $\sigma$ are $\sigma_i = p(i) \mod q$.
2. send $\sigma_i$ to $P_i$, for all $i$. 


3. publish auxiliary information allowing other players to check the shares are consistent and define a unique secret: \( \{ v_i = g^{s_i} \in \mathbb{G} \}_{i \in [t]} \) and \( v_0 = g^\sigma \in \mathbb{G} \).

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

\[
g^{s_i} = \prod_{j=0}^{t-1} v_j^{i_j} \in \mathbb{G}.
\]

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

**Commitments.** Our protocol uses non interactive equivocal commitments. An equivocable commitment scheme allows a sender \( S \) to commit to a message \( m \) s.t. \( S \)'s message is perfectly hidden; in the opening phase – where \( S \) reveals \( m \) and an opening value \( \text{d}(m) \) 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) \( \text{Setup}(1^\lambda) \rightarrow (\text{pp}, \text{tk}) \) which outputs public parameters \( \text{pp} \) and associated secret trapdoor key \( \text{tk} \); (2) \( \text{Com}(m, r) \rightarrow [\text{c}(m), \text{d}(m)] \) which on input a message \( m \) and random coins \( r \), outputs the commitment \( \text{c}(m) \) and an opening value \( \text{d}(m) \) (if \( S \) refuses to open a commitment \( \text{d}(m) = \bot \)); (3) \( \text{Open}(\text{c}, \text{d}) \rightarrow m \) or \( \bot \) which on input a commitment \( \text{c} \) and an opening value \( \text{d} \), outputs either a message \( m \) or an error symbol \( \bot \); (4) \( \text{Equiv}(\text{tk}, m, r, m') \rightarrow \tilde{d} \) which – if \( \text{tk} \) is a trapdoor key for \( \text{pp} \) – allows to open commitments \( \text{c}(m) \) to arbitrary values \( m' \). Precisely, for any messages \( m \) and \( m' \), any \( \text{Setup}(1^\lambda) \rightarrow (\text{pp}, \text{tk}) \), let \( \text{Com}(m, r) \rightarrow [\text{c}(m), \text{d}(m)] \) and \( \text{Equiv}(\text{tk}, m, r, m') \rightarrow \tilde{d} \) then \( \text{Open}(\text{c}(m), \tilde{d}) \rightarrow m' \); 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 \( [\text{c}(m), \text{d}(m)] \leftarrow \text{Com}(m, r) \), one has \( m \leftarrow \text{Open}(\text{c}(m), \text{d}(m)) \).

- **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 \( (\text{c}, \text{d}_0, \text{d}_1) \) s.t. \( \text{Open}(\text{c}, \text{d}_0) \rightarrow m_0; \text{Open}(\text{c}, \text{d}_1) \rightarrow m_1; m_0 \neq \bot; m_1 \neq \bot \) and \( m_0 \neq m_1 \) is negligible in the security parameter.

- **Concurrent non-malleability:** a commitment scheme is non-malleable \([\text{DDN00}]\) if no PPT adversary \( \mathcal{A} \) can “maul” a commitment to a value \( m \) into a commitment to a related value \( m' \). The notion of a concurrent non-malleable commitment \([\text{DDN00}] \text{PR05}\) further requires non-malleability to hold even if \( \mathcal{A} \) receives many commitments and can itself produce many commitments.

### 2.2 The elliptic curve digital signature algorithm

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

\[
\text{KeyGen}(\mathbb{G}, q, P) \rightarrow (x, Q) \text{ where } x \overset{\$}{\leftarrow} \mathbb{Z}/q\mathbb{Z} \text{ is the secret signing key and } Q := xP \text{ is the public verification key.}
\]

\[
\text{Sign}(x, m) \rightarrow (r, s) \text{ where } r \text{ and } s \text{ are computed as follows:}
\]

1. Compute \(m' := \mu\) leftmost bits of \(\text{SHA256}(m)\) where \(m\) is to be signed.
2. Sample \(k \overset{\$}{\leftarrow} \mathbb{Z}/q\mathbb{Z}\) 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 \(s := k \cdot (m' + r \cdot x) \mod q\).

\[
\text{Verif}(Q, m, (r, s)) \rightarrow \{0, 1\} \text{ indicating whether or not the signature is accepted.}
\]

As in \cite{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 \(A\), it gets to see the randomness \(R\) used to sign before choosing the message to be signed.

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

- \(\Theta^R\) to obtain a uniformly random point \(R = (r_x, r_y)\) in \(\mathbb{G}\);
- \(\Theta^{\text{Sign}(x, m; R)}\) which on input \(m \in \mathbb{Z}/q\mathbb{Z}\) chosen by \(A\), 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 \(\Theta^R\) but has not been previously used by \(\Theta^{\text{Sign}}\); else it returns \(\bot\).

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

Note that \(A\) chooses the messages queried to \(\Theta^\text{Sign}\), and knows (but does not choose) the randomness. Canetti et al. \cite{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:

\[
\text{lKeyGen}((\mathbb{G}, q, P); \ldots; (\mathbb{G}, q, P)) \rightarrow ((x_1, Q); \ldots; (x_n, Q)) \text{ s.t. } \text{KeyGen}(\mathbb{G}, q, P) \rightarrow (x, Q) \text{ where the values } x_1, \ldots, x_n \text{ constitute a } (t, n) \text{ threshold secret sharing of the signing key } x.
\]

\[
\text{lSign}((x_1, m); \ldots; (x_n, m)) \rightarrow ((r, s)) \text{ or } \bot \text{ where } \bot \text{ is the error output, signifying the parties may abort the protocol, and } \text{Sign}(x, m) \rightarrow (r, s).
\]
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 \(IS = (IKeyGen, ISign, \text{Verif})\), and a PPT algorithm \(A\), 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, \(A\) can chose these messages adaptively, and after seeing the randomness used in \(ISign\).

Let \(M\) be the set of aforementioned messages. The protocol \(IS\) is enhanced unforgeable if for any such \(A\), the probability \(\text{Adv}_{IS,A}^{e-tu-cma}\) that \(A\) can produce a valid signature on a message \(m \not\in M\) is a negligible function of \(\lambda\).

**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. To prevent such attacks, we provide sub-protocols (called identification protocols) allowing to identify the misbehaving party upon abort, and accordingly hold her accountable. This stronger model is referred to as secure multi-party computation with identifiable aborts (ID-MPC) \([IOZ14, CL14]\), and ensures that if the computation fails due to an abort, all honest parties learn the identity culprit, i.e. a corrupted party \(P_i\).

**On composability.** Our results are proven secure in the Random Oracle Model. We implicitly use the Fiat-Shamir transform to convert our honest-verifier sigma protocols into extractable non-interactive proofs. Extractability in the Fiat-Shamir paradigm relies on rewinding, which in the UC setting results in the running time of the simulator blowing up exponentially. One way to avoid the blow-up while allowing for extractability is to run proofs interactively, in which case they must be secure against malicious verifiers. Such an approach would lead to a large increase in the number of rounds of our protocol. Hence, motivated by efficiency, the definitions we adopt are game-based and do not guarantee composability.

### 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 \cite{CL15}, which was later enhanced in \cite{CLT18,CCL19} and gave a concrete instantiation from class groups of quadratic fields. Some background on class groups of quadratic fields in cryptography can be found in \cite{BH01} and in \cite{CL15} Appx. B.

We briefly sketch the instantiation given in \cite{CCL20,Sec. 2.3} and the resulting group generator \texttt{Gen} that we will use in this paper. The interested reader should refer to \cite{CL15,CCL19} for concrete details.

Given a prime $q$ consider another random prime $	ilde{q}$, the fundamental discriminant $\Delta_K = -qq$ and the associated class group $C(\Delta_K)$. By choosing $\tilde{q}$ s.t. $qq \equiv -1 \pmod{4}$ 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 $2^\lambda$. We then consider the suborder of discriminant $\Delta_q = -q^2\Delta_K$. Then, we denote $(\hat{G}, \cdot)$ 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 $\hat{G}$ (cf. \cite{Lag80}). One can exhibit a subgroup $F$ generated by $f \in \hat{G}$ 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. \cite{CL15} Proposition C – 1). Then we build deterministically a $q$–th power of $\hat{G}$ by lifting the class of an ideal of discriminant $\Delta_q$ 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. \cite{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 $2^\lambda$ running time and a $2^{-\lambda}$ success probability equates to $\lambda$ 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 ($2^\lambda$ 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 \texttt{Gen} the algorithm that on input a security parameter $\lambda$ and a prime $q$, outputs $(\tilde{s}, f, \hat{g}_q, \hat{G}, F)$ defined as above. We also denote \texttt{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. \cite{CCL19} 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 $G^q$ the subgroup consisting of all $q$–th powers in $G$,
and it’s order \( \hat{s} \). It holds that \( \hat{G} \) is the direct product of \( G^q \) and \( F \). We denote \( \varpi := \hat{s}_d \), the group exponent of \( G^q \), i.e. the least common multiple of the orders of its elements. Clearly, the order of any element in \( G^q \) divides \( \varpi \). In the following the distribution \( \mathcal{D} \) from which exponents are sampled is chosen to be close to uniform mod \( q \cdot \hat{s} \), where \( \hat{s} \) is an upper bound for \( \hat{s} \). This means that exponents sampled from \( \mathcal{D} \) follow a distribution close to uniform mod \( q \), and mod any divisor of \( \hat{s} \). In particular mod \( \varpi \).

**Hard subgroup membership assumption.** We recall the definition of the HSM problem for an output \( (\hat{s}, f, \hat{g}_q, \hat{G}, F) \) of Gen. For a random power \( g_q \) of \( \hat{g}_q \) the HSM assumption states it is hard to distinguish the elements of \( \lambda \) from a distribution close to uniform mod \( q \cdot \hat{s} \). In the following the distribution \( \mathcal{D} \) over the integers s.t. the distribution \( \{\text{int}_{\varpi}\} \) 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:

\[
Adv_{\mathcal{A}}^{\text{HSM}}(\lambda) := 2 \cdot \Pr[b = b^* : (\hat{s}, f, \hat{g}_q, \hat{G}, F) \leftarrow \text{Gen}(1^\lambda, q), t \leftarrow \mathcal{D}_q, g_q = \hat{g}_q^t, x \leftarrow \mathcal{D}, x' \leftarrow \mathcal{D}_q, b \leftarrow \{0, 1\}, Z_0 \leftarrow g_x^z, Z_1 \leftarrow g_{x'}^z, b^* \leftarrow \mathcal{A}(q, \hat{s}, f, \hat{g}_q, g_q, \hat{G}, F, Z_0, \text{Solve}(.)\} - 1
\]

The HSM problem is said to be hard in \( G \) if for all probabilistic polynomial time algorithm \( \mathcal{A} \), \( Adv_{\mathcal{A}}^{\text{HSM}}(\lambda) \) 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 \( \text{Gen}(1^\lambda, q) \) and as in Def. 3 we set \( g_q = \hat{g}_q^t \) for \( t \leftarrow \mathcal{D}_q \). The public parameters of the scheme are \( \text{pp}_{\text{CL}} := (\hat{s}, f, \hat{g}_q, \hat{G}, F, q) \).

To instantiate \( \mathcal{D}_q \), we set \( A > \hat{s} \cdot q \cdot 2^{40} \) s.t. \( \{g_q^r, r \leftarrow [A]\} \) 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. \( \{(g_q f)^r, r \leftarrow \mathcal{D}\} \) is at distance less than \( 2^{-\lambda} \) of the uniform distribution in \( G = F \times G^q \). The plaintext space is \( \mathbb{Z}/q\mathbb{Z} \). The scheme – hereafter referred to as the CL encryption scheme – is depicted in Fig. 1.

![Fig. 1: Description of the CL encryption scheme](image-url)

**Table:**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeyGen(pp_{\text{Cl}})</td>
<td>1. Pick ( sk \leftarrow \mathcal{D} ) and ( pk := g_q^sk ) 2. Return ( (pk, sk) )</td>
</tr>
<tr>
<td>Enc(pk, m)</td>
<td>1. Pick ( r \leftarrow [A] ) 2. Return ( (g_q^r, f^mpk^r) )</td>
</tr>
<tr>
<td>Dec(sk, (c_1, c_2))</td>
<td>1. Compute ( M = c_2/c_1^sk ) 2. Return ( \text{Solve}(M) )</td>
</tr>
</tbody>
</table>
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 \(\varpi\). The proof can be found in \[CCL+19\].

**Lemma 1.** Let \(\mathcal{D}\) be a distribution which is \(\delta\)-close to \(U(\mathbb{Z}/q\mathbb{Z})\). For any \(x \in G \setminus G^q\), \(\pi \leftarrow f^\gamma \in F\) where \(\gamma \leftarrow \mathbb{Z}/q\mathbb{Z}\) and \(k \leftarrow \mathcal{D}\), the distributions \(\mathcal{D}_1 := \{x, (k \mod \varpi), \pi \cdot x^k\}\) and \(\mathcal{D}_2 := \{x, (k \mod \varpi), x^k\}\) are \(2\delta\)-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 \(\delta_s\)-smooth if the distribution \(\mathcal{D}\) from which secret keys are sampled is \(\delta_s^2\)-close to \(U(\mathbb{Z}/q\mathbb{Z})\).

### 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 \textsc{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 \(\hat{G}\)), and a strong root assumption for class groups (stating that it is hard to find roots in \(\hat{G}\) 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 \textsc{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 \(\lambda \in \mathbb{N}\), and \(\gamma \in \mathbb{N}\). The \(\gamma\)-low order problem (\(\text{LOP}_\gamma\)) is \((t(\lambda), \epsilon_{\text{LO}}(\lambda))\)-secure for \(\text{Gen}\) if, given the output of \(\text{Gen}\), no algorithm \(\mathcal{A}\) running in time \(\leq t(\lambda)\) can output a \(\gamma\)-low order element in \(\hat{G}\) with probability greater than \(\epsilon_{\text{LO}}(\lambda)\). More precisely,

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

The \(\gamma\)-low order assumption holds if \(t = \text{poly}(\lambda)\), and \(\epsilon_{\text{LO}}\) is negligible in \(\lambda\).

**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 $\lambda \in \mathbb{N}$, and let $\mathcal{A}$ be a probabilistic algorithm. We run $\text{Gen}$ on input $(1^{\lambda}, q)$ to get $(s, f, \tilde{g}_q, \tilde{G}, F)$ and we give this output and a random $Y \in \langle \tilde{g}_q \rangle$ 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 $X \in \tilde{G}$, s.t. $Y = X^e$. In particular, the SRP is $(t(\lambda), \epsilon_{\text{SR}}(\lambda))$-secure for $\text{Gen}$ if any adversary $\mathcal{A}$, running in time $\leq t(\lambda)$, solves the SRP with probability at most $\epsilon_{\text{SR}}(\lambda)$.

## 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 $H \in \tilde{G}$. We then present the key generation sub-protocol for $\text{IKeyGen}$ and a key refresh sub-protocol. Next we provide our protocol for $\text{ISign}$, this is divided into two sub-protocols: an offline protocol $\text{Pre-Sign}$ allowing to pre-compute a number of pre-signatures before knowing the message to be signed; and an online protocol $\text{Sign}$ 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 ZK AoK we use to the hardness of the strong root assumption in $\tilde{G}$). Our $\text{ISetup}$ 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 $\text{pp}_G := (G, P, q)$ the description of the elliptic curve used in EC-DSA. For $n$ parties to collaboratively run $\text{ISetup}$, they proceed as depicted in Fig 2.

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

\[ \text{ISetup}(k, pp_G) \quad \text{All players } \{P_j\}_{j \neq i} \]

\[ r_i \overset{\$}{\leftarrow} \{0, 1\}^k \]
\[ [c_i, d_i] \leftarrow \text{Com}(r_i) \]
\[ \tilde{q} := \text{next-prime}(\bigoplus_{j=1}^n r_j) \]
\[ \text{Compute } \tilde{g}_q \text{ from } q, \tilde{q} \]
\[ t_i \overset{\$}{\leftarrow} [2^{40}] \text{ and } g_i \overset{\$}{\leftarrow} \tilde{g}_q^{t_i} \]
\[ h_i \overset{\$}{\leftarrow} \mathbb{Z}/q\mathbb{Z} \text{ and } H_i \leftarrow h_i \cdot P \]
\[ (\tilde{c}_i, \tilde{d}_i) \leftarrow \text{Com}((g_i, H_i)) \]
\[ \begin{align*}
\pi_i &:= \text{ZKPoK}_{g_i} \{(t_i, h_i) : g_i = \tilde{g}_q^{t_i} \land H_i = h_i \cdot P\} \\
&\text{if a proof fails abort} \\
g_q \leftarrow \prod_{j=1}^n \tilde{g}_q^{t_j} = \prod_{j=1}^n g_j \\
&\text{Erase all data other than } (g_q, H, pp_G)
\end{align*} \]

Fig. 2: Threshold CL setup used in IKeyGen

\[ P_i \]

\[ \text{IKeyGen}(g_q, H, pp_G) \quad \text{All players } \{P_j\}_{j \neq i} \]

\[ u_i \overset{\$}{\leftarrow} \mathbb{Z}/q\mathbb{Z} \text{ and } Q_i := u_i \cdot P \]
\[ [kgc_i, kgd_i] \leftarrow \text{Com}(Q_i) \]
\[ (s_k, pk_i) \leftarrow \text{CL.KeyGen}(1^\lambda) \]
\[ \begin{align*}
&\text{Perform } (t, n)\text{-VSS share of } u_i: \\
p_i(X) = u_i + \sum_{k=1}^t a_{i,k} X^k \mod q \\
&\text{Let } \{\sigma_{i,k} := p_i(j)\}_{j \in [n]} \text{ and } \{V_{i,k} := a_{i,k} P\}_{k \in [t]} \\
&\text{Send } \sigma_{i,j} \text{ to } P_j \overset{\$}{\leftarrow} \{V_{i,k}\}_{k \in [t]} \\
&\{\sigma_{i,k}\}_k \text{ are additive shares of } x_i := \sum_{k \in [n]} p_k(i) \\
&\text{where } \{x_i\}_{i \in [n]} \text{ are } (t, n) \text{ Shamir shares of } x. \\
\pi_i^{k_{ig}} &:= \text{ZKPoK}_{X_i} \{(x_i) : X_i = x_i P\} \\
&\text{Store } (Q, u_i, x_i)
\end{align*} \]

Fig. 3: Threshold Key Generation
**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 pp\(_G\) = \((G, P, q)\); the verification key \( Q \); and the public parameters pp\(_CL\) for the CL encryption scheme.

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 cannot obtain signatures on two different messages for the same randomness \( R \).

Furthermore, after each execution of Key Refresh, note that the shares \( u_i^{\text{new}} \) 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 IKeyGen.

<table>
<thead>
<tr>
<th>( P_i(u_i, Q_i, \text{pp}_G, \text{pp}_CL) )</th>
<th>Key Refresh</th>
<th>All players</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (sk_i, pk_i) \leftarrow \text{CL.KeyGen}(1^b) )</td>
<td>( sk_i \leftarrow pp )</td>
<td>( )</td>
</tr>
<tr>
<td>( v_{i,1}, \ldots, v_{i,n} \overset{$}{\leftarrow} \mathbb{Z}/q\mathbb{Z} ) s.t. ( \sum_j v_{i,j} = 0 )</td>
<td>( Y_i = {Q_{i,j} }<em>{j \in [n]} ) ( C</em>{i,j} := \text{Enc}(pk_j, v_{i,j}; \rho_j) ) ( \pi_{i,j}^v := \text{ZKAoK}<em>{\text{C}</em>{i,j}, Q_{i,j}}({v_{i,j}, \rho_j}) )</td>
<td>( {C_{i,j} }<em>{j \in [n]} \leftarrow Y_i ) if ( \sum</em>{j \in [n]} Q_{i,j} \neq 0 ) ( ) then abort</td>
</tr>
<tr>
<td>For ( j \in [n] ) let ( Q_{i,j} := v_{i,j} \cdot P )</td>
<td>( C_{i,j} = \text{Enc}(pk_j, v_{i,j}; \rho_j) \land Q_{i,j} = v_{i,j} \cdot P )</td>
<td>( {\pi_{i,j}^v }_{j \in [n]} \leftarrow ) if a proof fails abort</td>
</tr>
<tr>
<td>Sample ( \rho_j \overset{$}{\leftarrow} [\mathbb{A}] )</td>
<td>( )</td>
<td></td>
</tr>
<tr>
<td>( C_{i,j} )</td>
<td>( )</td>
<td></td>
</tr>
<tr>
<td>Overwrite old shares with:</td>
<td>( u_i^{\text{new}} := u_i + \sum_{j \in [n]} \text{Dec}(sk_i, C_{i,j}) \mod q )</td>
<td>( )</td>
</tr>
<tr>
<td>( Q_i^{\text{new}} := Q_i + \sum_{j \in [n]} Q_{i,j} )</td>
<td>( )</td>
<td></td>
</tr>
<tr>
<td>Erase previously computed pre-signatures and all Key Refresh data except ( u_i^{\text{new}}, Q_i, {pk_j }_{j \in [n]}, sk_i )</td>
<td>( )</td>
<td></td>
</tr>
</tbody>
</table>

**Fig. 4:** Key Refresh

**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 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 \( \gamma_i, w_i \in \mathbb{Z}/q\mathbb{Z} \) owned by \( P_i \) and \( k_j \in \mathbb{Z}/q\mathbb{Z} \) owned by \( P_j \), it allows to con-
vert multiplicative shares $k_j \gamma_i$ and $k_j w_i$ into additive shares $\alpha_{j,i}, \beta_{j,i}, \mu_{j,i}, \nu_{j,i} \in \mathbb{Z}/q\mathbb{Z}$ satisfying $\alpha_{j,i} + \beta_{j,i} = k_j \gamma_i \mod q$ and $\mu_{j,i} + \nu_{j,i} = k_j w_i \mod q$. 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.

### 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 identify 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. To ensure this, a player must prove that the ciphertext it first sent decrypts to some message or to nothing (this happens if the ciphertext is not well-formed). To this end, we provide an efficient zero-knowledge argument of knowledge of some information related to the secret key used to decrypt a CL ciphertext. The proof we provide is specific to our considered encryption scheme, precisely, it is for the following relation:

$$R_{\text{Dec}} := \{(pk, (c_1, c_2), M); (\sigma_0, \sigma_1); | c_1, c_2, M \in G; \sigma_0 \in \mathbb{N},$$

$$\sigma_1 \in [-\overline{AC}(2^{40} + 1) \cdot 2^{\sigma_0}, \overline{AC}(2^{40} + 1) \cdot 2^{\sigma_0}]; c_2 \cdot M^{-1} = c_1^{2^{\sigma_0} \sigma_1} \land pk = g_q^{2^{\sigma_0} \sigma_1}\}. $$

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 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 ⊥ if decryption fails) in another way.

**Arguing knowledge of a decrypted message.** We show here how a proof for $R_{\text{Dec}}$ works. Consider a party $P$ with encryption key pair $(pk, sk)$ and suppose $P$ has received a ciphertext $\vec{c} = (c_1, c_2) \in G^2$, encrypted under $pk$. The following
Fig. 5: Pre-Sign: Offline Threshold Pre-Signature Protocol
proof allows $P$ to convince a verifier that decrypting $\vec{c}$ with some information of the secret key ($\text{sk}$ if $P$ is honest) yields $m$, which may either be $\perp$ if $\vec{c}$ fails to decrypt, or some value in $\mathbb{Z}/q\mathbb{Z}$.

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 $M := c_2 \cdot \vec{c}^{-\text{sk}}$. Note that if $M = f^a$ for some $a \in \mathbb{Z}/q\mathbb{Z}$, the decryption algorithm returns $a$, otherwise it returns $\perp$. 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:

$$\mathbb{R}_{\text{Dec}} := \{(pk, (c_1, c_2, M); (\sigma_0, \sigma_1); | c_1, c_2, M \in \hat{G}; \sigma_0 \in \mathbb{N}, \sigma_1 \in [-\hat{A} C(2^{40} + 1) \cdot 2^\sigma_0, \hat{A} C(2^{40} + 1) \cdot 2^\sigma_0]; c_2 \cdot M^{-1} = c_1^{2^\sigma_0 \sigma_1} \land pk = g_q^{2^\sigma_0 \sigma_1}\}.$$

This proof does not allow the extraction of an integer value of the secret key, however it is enough to prove that $pk \in \langle g_q \rangle$ and that the exponent of $c_1$ w.r.t. to $c_2 \cdot M^{-1}$ and of $pk$ in base $g_q$ is the same when the decryption is $m \neq \perp$. Furthermore an honest prover knows and uses its secret key in the protocol.

The interactive protocol is given in Fig. 7. We denote $C$ the challenge set, and $C := |C|$. The only constraint on $C$ is that the $C$-low order assumption holds in $\hat{G}$. The protocol is complete, honest verifier zero-knowledge, and sound under the assumption the strong root problem for class groups with input $(\hat{G}, G^q, g_q)$, and the $C$-low order problem in $\hat{G}$ are hard. The proof is essentially that of [CCL+20, Theorem 2] for their ZKAoK for relation $\mathbb{R}_{\text{Enc}}$ with very minor variations. More formally, the following theorem is valid for $\mathbb{R}_{\text{Dec}}$.

**Theorem 1.** If the strong root assumption is $(t'(\lambda), \epsilon_{SR}(\lambda))$-secure for Gen, and the $C$-low order assumption is $(t'(\lambda), \epsilon_{LO}(\lambda))$-secure for Gen, denoting $\epsilon := \max(\epsilon_{SR}(\lambda), \epsilon_{LO}(\lambda))$, then the interactive protocol of Fig. 7 is a computationally convincing proof of knowledge for $\mathbb{R}_{\text{Dec}}$ with knowledge error $\kappa(\lambda)$, time bound $t(\lambda)$ and failure probability $\nu(\lambda)$, where $\nu(\lambda) = 8\epsilon$, $t(\lambda) < t'(\lambda)/448$ and $\kappa(\lambda) = \max(4/C, 448t(\lambda)/t'(\lambda))$. If $r \in [3 \cdot 2^{40}]$ (it is so when the prover is honest), the protocol is honest verifier statistical zero-knowledge.

**Other zero-knowledge for relations in CL.** Our proof for $\mathbb{R}_{\text{Dec}}$ is similar to the ZKAoK for $\mathbb{R}_{\text{Enc}}$ relation presented in [CCL+20]. We also use a proof for $\mathbb{R}_{\text{Enc}}$ in this work, which is $\pi_i$ in Fig. 5. Phase 1. For completeness, we recall

<table>
<thead>
<tr>
<th>$P_i$</th>
<th>Phase 7</th>
<th>All players ${P_j}_{j \neq i}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i := mk_i + r\sigma_i$</td>
<td>$\underset{\sim}{s_i}$</td>
<td>$s := \sum_{i \in S} s_i$.</td>
</tr>
<tr>
<td>Erase $(R, k_1, \sigma_i)$</td>
<td>if $(r, s)$ is not a valid signature, <strong>abort</strong>, else return $(r, s)$.</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 6: Sign: Online Threshold Signature Protocol
Setup:

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

### Prover (pk, \((c_1, c_2), M; sk\))

- \(r \xleftarrow\$ [2^{40}\tilde{A}C]\)
- \(t_1 := c_1^r \) and \(t_2 := g_q^r\)

### Verifier (pk, \((c_1, c_2), M\))

- \(k \xleftarrow\$ C\)
- \(u := r + k \cdot sk \in \mathbb{Z}\)
- \(\text{Check } u \in [\tilde{A}(2^{40} + 1)]\)
- \(c_1^r = t_1 \cdot (c_2 \cdot M^{-1})^k\)
- \(g_q^r = t_2 \cdot pk^k\)

Fig. 7: Zero-knowledge argument of knowledge for \(R_{\text{Dec}}\).

below the \(R_{\text{Enc}}\) relation. Consider a prover \(P\) having computed an encryption of \(a \in \mathbb{Z}/q\mathbb{Z}\) with randomness \(r \xleftarrow\$ \tilde{A}\), i.e. \(c := (c_1, c_2)\) with \(c_1 := g_q^r, c_2 := pk^rf^a\), the \(R_{\text{Enc}}\) relation is defined as:

\[
R_{\text{Enc}} := \{(pk, c); (a, (\rho_0, \rho_1)) \mid pk \in \hat{G}; a \in \mathbb{Z}/q\mathbb{Z}; \rho_0 \in \mathbb{N}, \\
\rho_1 \in [-\tilde{A}(2^{40} + 1) \cdot 2^{\rho_0}, \tilde{A}(2^{40} + 1) \cdot 2^{\rho_0}]; c_1 = g_q^{2^{\rho_0} \cdot \rho_1} \land c_2 = pk^{2^{\rho_0} \cdot \rho_1} f^a\}.
\]

Furthermore, we also use a ZK AoK for the relation \(R_{\text{CL-DL}}\) defined as follows. Consider a prover having computed an encryption of \(a \in \mathbb{Z}/q\mathbb{Z}\) with randomness \(r \xleftarrow\$ \tilde{A}\), i.e. \(c := (c_1, c_2)\) with \(c_1 := g_q^r, c_2 := pk^rf^a\) and point \(Q\) of the curve from a random public point \(P\) as \(Q = aP\), the \(R_{\text{CL-DL}}\) is defined as:

\[
R_{\text{CL-DL}} := \{(pk, (c_1, c_2), Q); (a, r) \mid c_1 = g_q^r \land c_2 = pk^rf^a \land Q = aP\}.
\]

A ZK PoK for \(R_{\text{CL-DL}}\) was presented in [CCL+19] and improved in [CCL+20]. Furthermore, in [CCL+20], the authors presented a very efficient zero-knowledge argument of knowledge for a slightly different relation \(\bar{R}_{\text{CL-DL}}\), which is the proof we use in this work. Relation \(\bar{R}_{\text{CL-DL}}\) is defined as:

\[
\bar{R}_{\text{CL-DL}} := \{(pk, (c_1, c_2), Q); (a, (\rho_0, \rho_1)) \mid c_1 = g_q^{2^{\rho_0} \cdot \rho_1} \land c_2 = f^a pk^{2^{\rho_0} \cdot \rho_1} \land Q = aP\}.
\]

Proofs for \(\bar{R}_{\text{CL-DL}}\) in our protocol are \(\pi_i^j\) in Fig. 5 Phase 5 and \(\pi_i^{k_j}\) in Fig. 4.

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 $\pi_{kg}^i$ 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 $0G$ or a proof $\pi_{kr}^{i,j}$ 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 ($\alpha$ or $\mu$).
2. Phase 2. If the check on $\mu$ fails.
3. Phase 5. If $P \neq \sum \hat{R}_i$.
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 [3] if the protocol reached this point, then $P = \sum \hat{R}_i$ 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 $s_i \cdot R \neq m \cdot \hat{R}_i + r \cdot S_i$. 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 $k_i, \gamma_i$ 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 $\gamma_i$ 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 $M := c_2 \cdot c_1^{-sk_j}$, and proves that $(pk_j, (c_1, c_2), M) \in R_{Dec}$, using the proof of Subsection [1]. 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 $(pk_j, c_{k_j} = (c_{k_j,1}, c_{k_j,2}), f_{k_j}) \in R_{Dec}$, 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 $(pk_j, (c_1, c_2), f_{\mu_{j,i}}) \in R_{Dec}$, using the proof of Subsection [3]. 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 $P \neq \sum R_i$ 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 $\{\pi_{kg,i}\}_{i \in [n]}$ and $\{\pi_i, \pi_i', \pi_i''\}_{i \in S}$ have been accepted.

The identification protocol uses new techniques, and requires the use of a zero-knowledge proof for $R_{Dec}$ to prove ciphertexts decrypt to a given value (e.g. the proof of Subsection [4]). 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 $a_{sk}$ in $IKeyGen$, this item holds.
(iii) The value $\tilde{\gamma}_i$ input to the MtA protocol is consistent with $\Gamma_i = \gamma_i \cdot P$ that is decommitted to in Phase 4, i.e. $\gamma_i = \tilde{\gamma}_i$.

(iv) The value $\delta_i$ 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 $\gamma_j$. All players then check that $\Gamma_j$ revealed in Phase 4 satisfies $\Gamma_j = \gamma_j \cdot P$. If the check fails, $P_j$ is identified as a cheater.

(b) After all $\beta_{i,j}$ and $\gamma_i$, for $i \neq j$ and for $j \neq i$ have been published, $P_j$ reveals $k_j$ and performs a ZK proof that $(pk_j, c_k, f^{k_j}) \in R_{Dec}$. If the proof is accepted, all parties are convinced $c_k$ decrypts to $k_j$ using $P_j$’s decryption key; else $P_j$ is identified as a cheater.

(c) For $i \neq j$, reveal $\alpha_{i,j}$ and $c_k, \gamma_i$. Then perform a ZK proof that $(pk_j, c_k, \gamma_i, f^{\alpha_{i,j}}) \in R_{Dec}$. If the proof is accepted, all parties are convinced $c_k, \gamma_i$ decrypts to $\alpha_{i,j}$ using $P_j$’s decryption key; else $P_j$ is identified as a cheater.

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

$$\tilde{\beta}_{i,j} := k_i \gamma_j - \alpha_{i,j} \quad \text{and check if } \tilde{\beta}_{i,j} = \tilde{\beta}_{i,j}. \quad \text{If it is not true for some index } i, \text{ then } P_j \text{ is the cheater (for using } \tilde{\gamma}_j \neq \gamma_j \text{ in the MtA protocol).}$$

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 $\gamma_j$ published in step (a) satisfying $\Gamma_j = \gamma_j \cdot P$, then to have $\beta_{i,j} = \tilde{\beta}_{i,j}$, 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 $\mathbb{Z}/q\mathbb{Z}$ 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 $\tilde{\beta}_{i,j} = k_i \gamma_j - \alpha_{i,j}$ with probability significantly greater than $1/q$. 

**(iv)** Now if consistency of (iii) holds, all parties can check that $\delta_j = k_j \gamma_j + \sum_{j \neq i} \alpha_{j,i} + \sum_{i \neq j} \beta_{i,j}$; if not $P_j$ is identified as the cheater. If equality holds then consistency of (iv) holds (i.e. consistency of $\delta_j$).
Identification - Abort of type \( \text{\texttt{4}} \) in Phase \( \text{\texttt{6}} \). Since this failure occurs in Phase \( \text{\texttt{6}} \), we know that consistency of \((\text{\texttt{i}}), (\text{\texttt{ii}}), (\text{\texttt{iii}})\) and \((\text{\texttt{iv}})\) hold. The abort must hence be due to \((\text{\texttt{v}})\) not holding. To detect a cheater, each \( P_j \) does the following:

1. Publish \( k_j \) and perform a ZK proof that \((\text{pk}_j, c_{k_j}, f_{k_j}) \in \text{R}_{\text{Dec}} \). 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 \( c_{k_j} \cdot w_i \). Perform a ZK proof that \((\text{pk}_j, c_{k_j} \cdot w_i, f_{\mu_{j,i}}) \in \text{R}_{\text{Dec}} \). If the proof is accepted, all parties are convinced \( c_{k_j} \cdot w_i \) 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 \( \Sigma_j \) is equal to the discrete log in base \( R \) of \( S_j \). Precisely, let for public parameters \((G, P, q)\), let \( R_{\log} := \{ (R, S, T) \in G^3; \sigma | R = \sigma \cdot S \land T = \sigma \cdot P \} \). Then \( P_j \) performs a ZKPoK \( \pi_{\log}^j \) that \((R, S_j, \Sigma_j) \in R_{\log} \); 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 \((\text{\texttt{v}})\) 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, $S$ gets as input an EC-DSA public key $Q$, where $Q = x \cdot P$, and can query oracles $O^R$ and $O^{\text{Sign}(x, \cdot : \cdot)}$ of Def 1. After this query phase, $S$ 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 $A$ corrupts $\{P_j\}_{j > 1}$, one can construct $S$ simulating $P_1$ s.t. the output distribution of $S$ is indistinguishable from $A$’s view in an interaction with an honest party $P_1$.

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

The main differences with the proof of [GG20], arises from the fact $S$ 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 $S$ 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, $S$ 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 $S$:**

1. $S$ receives a public key $Q$ from it’s EC-DSA challenger.
2. $S$ samples a CL encryption key pair $(pk_1, sk_1) \leftarrow \text{KeyGen}(1^\lambda)$, and a random value $u_1 \in \mathbb{Z}/q\mathbb{Z}$. It computes $[kgc_1, kgd_1] \leftarrow \text{Com}(u_1 P)$ and broadcasts $pk_1$ and $kgc_1$. In return, $S$ receives the public keys $\{pk_j\}_{j \in [n], j \neq 1}$ and commitments $\{kgc_j\}_{j \in [n], j \neq 1}$.
3. $S$ broadcasts $kgd_1$ and receives $\{kgd_j\}_{j \in [n], j \neq 1}$. For $i \in [n]$, let $Q_i \leftarrow \text{Open}(kgc_i, kgd_i)$ be the revealed commitment value of each party. Each player performs a $(t, n)$ Feldman-VSS of the value $Q_i$, with $Q_i$ as the free term in the exponent.
4. $S$ rewinds $A$ to the decommitment step and
   - equivocates $P_1$’s commitment to $\overline{kgd}$ so that the committed value revealed is now $\overline{Q}_1 := Q - \sum_{j=2}^{n} Q_j$,
   - simulates the Feldman-VSS with free term $\overline{Q}_1$.
5. \( \mathcal{A} \) will broadcast the decommitments \( \{k_\text{gd}_j\}_{j \in [n], j \neq 1} \). Let \( \{\tilde{Q}_j\}_{j=2 \ldots n} \) be the committed value revealed by \( \mathcal{A} \) at this point (this could be \( \perp \) if \( \mathcal{A} \) refuses to decommit).

6. All players compute the public signing key \( \tilde{Q} := \sum_{i=1}^{n} \tilde{Q}_i \). If any \( \tilde{Q}_i = \perp \), then \( \tilde{Q} := \perp \).

7. 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.

8. \( \delta \) simulates \( \pi_{k_\text{gd}, 1} \) (the ZKPoK that it knows \( x_1 \) corresponding to \( X_1 \)). Then, for \( j \in [n], j \neq 1 \), \( \delta \) receives from \( \mathcal{A} \) a ZKPoK of \( x_j \) satisfying \( X_j := x_j \cdot P \); from which \( \delta \) can extract \( x_j \).

**Simulating the Key Refresh Protocol.** For all honest players \( P_i \), \( \delta \) 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 \( \Theta^R \) to obtain a uniformly random point \( R = (r_x, r_y) \) in \( G \):
- oracle \( \Theta^\text{Sign}(\mathcal{A}, m; R) \) which on input a message \( m \) chosen by \( \mathcal{A} \), returns a valid signature \( (r, s) \) for \( m \) where \( r := r_x \mod q \) if \( R = (r_x, r_y) \) was queried to \( \Theta^R \); 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 \( \tilde{Q} \) for the online phase (Phase 7). We stress that though the simulator knows the decryption key \( \text{sk}_1 \), 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 \( \delta \) 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 \( \delta \):**

**Phase 1:** As in a real execution, \( \delta \) samples \( k_1, \gamma_1 \leftarrow \mathbb{Z}/q\mathbb{Z}, r_1 \leftarrow [\tilde{A}] \) uniformly at random. It computes \( c_{k_1} \leftarrow \text{Enc}(\text{pk}_1, k_1; r_1) \), the associated ZK AoK \( \pi_1 \), and \( [c_1, d_1] \leftarrow \text{Com}(\gamma_1 P) \). It broadcasts \( c_1, c_{k_1}, \pi_1 \) before receiving \( \{c_j, c_{k_j}, \pi_j\}_{j \in S, j \neq 1} \) from \( \mathcal{A} \). \( \delta \) 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 \( P_j, j \in S, j \neq 1 \). \( \delta \) runs the protocol for \( P_1 \) as follows:
(a) *Initiator for MtA with $k_1$ and $\gamma_1$: Since $S$ knows $k_1$, it runs the protocol as would an honest $P_1$; it also decrypts $c_{k_1,\gamma_1}$ received from $P_j$ thereby obtaining $\alpha_{1,j}$ mod $q$.

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

(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 $S$ 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 $\mu_{1,j}P + B_{1,j} = k_1W_j$. Furthermore, recall that $S$ extracted $x_j$ from $\pi_{kj,j}$ in KeyGen, hence it knows $w_j$, so $S$ can compute $\nu_{1,j} = k_1w_j - \mu_{1,j}$ mod $q$.

(d) *Respondent for MtAwc with $k_j$ and $w_1$: Here, $S$ only knows $W_1 = w_1 \cdot P$ (but not $w_1$). So it samples a random $\mu_{j,1} \leftarrow \mathbb{Z}/q \mathbb{Z}$ and sets $c_{k_1,w_1} \leftarrow \text{Enc(pk}_j, \mu_{1,1})$, and $B_{1,j} := k_j \cdot W_1 - \mu_{j,1} \cdot P$. Finally $S$ sends the ciphertexts and the point.

Note that at this point $S$ 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 $S$; $\beta_{j,1}$ is chosen by $S$; and $\alpha_{j,1} = k_j \gamma_1 - \beta_{j,1}$, where $S$ 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, $\mu_{j,1}, \nu_{j,1} \cdot P = B_{j,1}, j \in S, j \neq 1$ as respondent for MtAwc. Indeed, $\mu_{1,j}$ is a value decrypted by $S$; $\mu_{j,1}$ is chosen by $S$ in (d); $\nu_{1,j} = k_1w_j - \mu_{1,j}$, where $S$ knows all the values on the right.

After all these sub-protocols, $S$ 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 $S$ does not know the internal values from the MtA and MtAwc protocols executed by two players that are both controlled by the adversary. Hence $S$ is not able to compute the values $\sigma_j$ and $\delta_j$ for $j \in S, j \neq 1$. Furthermore $S$ cannot compute $\sigma_1$ since it doesn’t know the value $w_1$, but it can compute

$$\sigma_C := \sum_{i > 1} \sigma_i = \sum_{i > 1} (k_iw_i + \sum_{j \neq i} \mu_{i,j} + \sum_{j \neq i} \nu_{j,i}) = \sum_{i > 1} \sum_{j \neq i} (\mu_{i,j} + \nu_{j,i}) + \sum_{i > 1} k_iw_i$$

$$= \sum_{i > 1} (\mu_{i,1} + \nu_{i,1}) + \sum_{i > 1} k_iw_i$$

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 $\nu_{1,j} = k_1w_j - \mu_{1,j}$ mod $q$.

Furthermore, as $S$ knows $W_1$, it can compute $\Sigma_1 := \sigma_1 \cdot P = k_1W_1 + \sum_{j \neq 1}(\mu_{1,j} + \nu_{j,1})P$.  

25
Phase 3: \( \delta \) broadcasts \( \delta_1 \) and receives all the \( \{\delta_j\}_{j \in S, j \neq 1} \) from \( \mathcal{A} \). Let \( \bar{\delta} := \sum_{i \in S} \delta_i \).

Next \( \delta \) samples a random \( \ell_1 \sim \mathbb{Z}/q \mathbb{Z} \) and broadcasts \( T_1 := \Sigma_1 + \ell_1 \cdot H \).

Note that \( \delta \) does not know \( \sigma_1 \), so it simulates the ZK proof \( \bar{\pi}_1 \). Then, from the proofs \( \bar{\pi}_j \) received from \( \mathcal{A} \), \( \delta \) extracts the values \( (\bar{\sigma}_j, \ell_j) \) for \( j \in S, j \neq 1 \).

We hereafter denote \( \bar{\sigma}_C := \sum_{j \in S, j \neq 1} \bar{\sigma}_j \).

Phase 4: \( \delta \) broadcasts \( d_1 \) which decommits to \( \Gamma_1 \), and \( \mathcal{A} \) reveals \( \{d_j\}_{j \in S, j \neq 1} \) which decommit to \( \{\Gamma_j\}_{j \in S, j \neq 1} \).

From the proofs \( \pi_j \) on \( \Gamma_j \), \( \delta \) can extract \( \gamma_j \) 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 \( \delta \) 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 \( \bar{\gamma}_i \) in the MtA protocol than the \( \gamma_i \) committed to in Phase 1, hence we denote them with a tilde.

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

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

Then using the values \( k_i \) extracted in Phase 1, \( \delta \) checks that \( \sum_i k_i \cdot \bar{R} = P \).

Notice that if

\[
\sum_i k_i \cdot \bar{R} = k \cdot \bar{R} = k \bar{\delta}^{-1} \sum_i \gamma_i \cdot P = k \bar{\delta}^{-1} \gamma \cdot P = P \Rightarrow \bar{\delta} = k \gamma = \delta
\]

The simulator can also detect if the values \( \sigma_j \) computed in Phase 2 are consistent with those used to compute points \( \ell_j \) in Phase 3; in particular \( \delta \) checks that \( \bar{\sigma}_C = \sigma_C \). We thus distinguish two types of executions: an execution is said to be semi-correct if

\[
\sum_{i} k_i \bar{R} = P \quad \text{and} \quad \bar{\sigma}_C = \sigma_C
\]

which, as explained above, implies that \( \delta = \bar{\delta} \) and \( \bar{\sigma}_C = \sigma_C \). If either of the above equalities do not hold, the execution is said to be non semi-correct.

Now \( \delta \) adapts its behaviour depending on the type of execution:

- **Semi-correct execution:**
  1. \( \delta \) invokes oracle \( \mathcal{O}^R \) to obtain \( R = (r_x, r_y) \).
  2. \( \delta \) sets \( \bar{\Gamma}_1 := \bar{\delta} \cdot R - \sum_{i \in S, i \neq 1} \Gamma_i \), so that \( R = \bar{\delta}^{-1} \left( \bar{\Gamma}_1 + \sum_{i \in S, i \neq 1} \Gamma_i \right) \).

Then \( \delta \) rewinds \( \mathcal{A} \) to the decommitment step in Phase 4, equivocates \( P_1 \)'s commitment so that it decommits to \( \bar{\Gamma}_1 \) instead of \( \Gamma_1 \).

- **Non semi-correct execution:** \( \delta \) simply moves on to Phase 5.

Phase 5: **Semi-correct execution:** \( \delta \) publishes \( \bar{R}_1 := P - \sum_{i \in S, i \neq 1} k_i \cdot R \) together with \( \pi_1' \); a simulated ZKP of consistency with \( e_{k_1} = \text{Enc}_1(k_1) \) (note that in this case \( \bar{R}_1 \neq k_1 \cdot R \) due to the rewinding).
• **Non semi-correct execution**: $\mathcal{S}$ publishes $\mathcal{R}_1 := k_1 \cdot R$ together with $\pi'_j$: a real ZKP of consistency with $c_{k_1} = \text{Enc}_1(k_1)$ (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 $\pi''_j$: 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 $\pi''_j$: 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 adversary’s proofs $\pi'_j$ or $\pi''_j$ for some $j > 1$ will fail, and the protocol will abort.

**Phase 7**: $\mathcal{S}$ invokes the second oracle $O^{\text{Sign}}(sk, m; R)$ with input $m$. In return, $\mathcal{S}$ receives the valid signature $(r, s)$ on $m$, where $r = r_x \mod q$, 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 $\sigma_C$ is as defined in the simulation of Phase 2 (Note: if $\mathcal{A}$ cheats in Phase 7 – denoting $\{s_j\}_{i > 1}$ the values $\mathcal{S}$ receives from $\mathcal{A}$ in Phase 7, and $s_C := \sum_{i > 1} s_i$ – it is possible that $s_C \neq s_C$). 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 10:

- 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 $R_{\text{Dec}}$ 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 $\mu_{1,j}$ it received, observe that: if $\mu_{1,j}$ is the real decryption of $c_{k_j,w_1}$ (which it must be if the proof for $R_{\text{Dec}}$ 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 21).
Abort of type $4$: $\delta$ follows the real identification procedure, up until it needs to prove knowledge of $\sigma_1$ satisfying $S_1 = \sigma_1 \cdot R$. Since $\delta$ does not know $\sigma_1$, it simulates the proof $\pi^{\log_1}$.

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, $\delta$ 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 $A$ 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 $\delta$ 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 $s_C \neq \tilde{s}_C$, $\delta$ 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 $\delta$ 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 $A$ 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]. Furthermore the simulator runs the key refresh sub-protocol as in a real execution, hence the simulation there is perfect.

**Lemma 2.** Assuming the strong root assumption and the $C$-low order assumption hold for $Gen$; the CL encryption scheme is $\delta_s$-smooth; the HSM problem is $\delta_{HSM}$-hard; 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 $\delta$ now also simulates the identification procedure when the protocol aborts.

Before proving Lemma 2, let us define the notion of invalid ciphertexts.

**Definition 7.** A ciphertext is said to be invalid if it is of the form $(u, e) := (u, u^{sk} f^m)$ where $u \in G \setminus G^\alpha$. 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 pre-signing and signing protocols. The differences between $\mathcal{A}$’s real and simulated view are the following:

1. $\delta$ does not know $w_1$. So for each $j \in S, j \neq 1$ it cannot compute $c_{k,j_1,w_1}$ as in a real execution of the protocol. However under the strong root and $C$-low order assumption in $G$, $\delta$ can extract $k_j$ from proof $\pi_j$ in Phase 1 for each $j \in S, j \neq 1$. $\delta$ needs to simulate $P_1$ as a respondent in MtA 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 $\mathbb{Z}/q\mathbb{Z}$. Moreover $c_{k_1}$ was proven to be a valid ciphertext, so ciphertexts computed using homomorphic operations over $c_{k_1}$ and fresh ciphertexts computed with $pk_1$ follow identical distributions from $\mathcal{A}$’s view.

2. $\delta$ computes $\hat{\Gamma}_1 := \delta \cdot R - \sum_{i \in S, i \neq 1} \Gamma_i$, and equivocates its commitment $c_1$ s.t. $d_1$ decommits to $\hat{\Gamma}_1$. Let us use $\tilde{\gamma}_1 \in \mathbb{Z}/q\mathbb{Z}$ the value s.t. $\tilde{\Gamma}_1 = \tilde{\gamma}_1 P$, where $\tilde{\gamma}_1$ is unknown to $\delta$, but the forger can simulate the ZKPoK of $\tilde{\gamma}_1$.

Let us further denote $k \in \mathbb{Z}/q\mathbb{Z}$ the randomness (unknown to $\delta$) used by its’ signing oracle to produce $R$. It holds that $\delta = k(\tilde{\gamma}_1 + \sum_{j \in S, j \neq 1} \gamma_j)$, where $\Gamma_i = \gamma_i \cdot P$, to distinguish them from the $\tilde{\gamma}_i$s which are used in MtA protocols. Finally, let us denote $\tilde{k}_1 := \tilde{k} - \sum_{j \in S, j \neq 1} k_j$. $\delta$ is implicitly using $\tilde{k}_1 \neq k_1$, 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 $\delta_s$-smooth and the HSM problem is $\delta_{HSM}$-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 $\tilde{k}$ with probability greater than $2\delta_{HSM} + 3/q + 4\delta_s$.

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

**Game0 to Game1.** $\delta$ uses the secret key $sk_1$ instead of the public key $pk_1$ and $r_1$ to compute $c_{k_1} \leftarrow (u_1, u_1^{sk_1} f_{k_1})$ where $u_1 = g_q^{\ast}$. The simulation of honest players uses the public key as usual. Both games are perfectly indistinguishable from $\mathcal{A}$’s view:

$$|\Pr[E_1] - \Pr[E_0]| = 0.$$

**Game1 to Game2.** In Game2 one replaces the first element of $c_{k_1}$ (in Game1 this is $u_1 \in G^q$) with $\tilde{u}_1 \in G \setminus G^q$. There exists a unique $r_1 \in \mathbb{Z}/s\mathbb{Z}$ and $b_1 \in \mathbb{Z}/q\mathbb{Z}$ such that $\tilde{u}_1 = g_q^{\ast} f_{b_1}$. And $c_{k_1} = (\tilde{u}_1, \tilde{u}_1^{sk_1} f_{k_1})$. Under the $\delta_{HSM}$-hardness of HSM both games are indistinguishable:

$$|\Pr[E_2] - \Pr[E_1]| \leq \delta_{HSM}.$$

**Game2 to Game3.** In Game3 the points $Q = x \cdot P$ and $R = \tilde{k}^{-1} \cdot P$ come from the EC-DSA oracle, while in Game2 they are computed as in the real
The smoothness of the CL cannot distinguish distribution modulo q are δ-close. Indeed, A’s view of α’s view differs when S uses α, μ in Game3 from when it uses α’, μ’ in Game3’ is negligible, we can conclude that A cannot distinguish Game2 and Game3’ except with negligible probability. The smoothness of the CL encryption scheme tells us that given pk1, which fixes (sk1 mod s), the value of (sk1 mod q) remains δ-close to the uniform distribution modulo q. In particular this ensures that A’s view of α and α’ are δ-close. Indeed, A receives an invalid encryption of k1, which information theoretically masks k1. At this point A’s view of k1 is that of a random variable δ-close to the uniform distribution modulo q. A then computes cα
which it sends to $S$. Finally $A$ 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 $\mu$ and $\mu'$, the indistinguishability of $A$'s view of both random variables is a little more delicate, since $A$ gets additional information from the check on the curve performed by $S$, namely in Games 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 Games', if $\mu' \cdot P + B_{1,j} \neq \tilde{k}_1 \cdot W_j$ the simulator aborts. We call the output of this check test'. Notice that if test = test', both games are $\delta_s$-close from $A$'s view (the only change is in the ciphertext $c_{\mu'}$). Let us bound the probability $p$ that test $\neq$ test'. This will allow us to conclude that

$$|\Pr[E_3] - \Pr[E_2]| \leq p + \delta_s.$$  

Let us consider the ciphertext $c_{\mu} = (u_{\mu}, e_{\mu}) \in \hat{G} \times \hat{G}$ sent by $A$. There exist unique $z_{\mu} \in \hat{G}$, $y_{\mu} \in F$ such that $u_{\mu} = z_{\mu}y_{\mu}$. Moreover there exists a unique $b_{\mu} \in Z/qZ$ such that $y_{\mu} = f^{b_{\mu}}$. Since $sk_1 = \tilde{sk}_1$ or $\mu = \perp$ if and only if $\mu' = \perp$, and this occurs when $e_{\mu} \cdot z_{\mu}^{-sk_1} = e_{\mu'} \cdot z_{\mu}^{-\tilde{sk}_1} \notin F$. In this case Games is identical to Games' from $A$'s view ($S$ aborts in both cases). We hereafter assume decryption does not fail, which allows us to adopt the following notation $c_{\mu} = z_{\mu}^{sk_1} f^{b_{\mu}} = z_{\mu}^{\tilde{sk}_1} f^{h_{\mu}}$ with $h_{\mu} \in Z/qZ$. We thus have:

$$\mu := \log f \left( \frac{e_{\mu}}{z_{\mu}^{sk_1}} \right), \quad \mu' := \log f \left( \frac{e_{\mu}}{z_{\mu}^{\tilde{sk}_1}} \right),$$

Thus

$$= h_{\mu} - b_{\mu} sk_1 \mod q \quad = h_{\mu} - b_{\mu} \tilde{sk}_1 \mod q$$

we have

$$\mu - \mu' \equiv b_{\mu} (\tilde{sk}_1 - sk_1) \equiv b_{\mu} b_1^{-1} (k_1 - \tilde{k}_1) \mod q.$$  

We consider three cases:

(a) $\mu = \mu'$ mod $q$. This may happen for two reasons:

i. If $k_1 \equiv \tilde{k}_1$ mod $q$, then Games and Games are identical.

ii. Else $b_{\mu} = 0$ mod $q$, i.e. $c_{\mu}$ is a valid ciphertext. Since we ruled out $k_1 \equiv \tilde{k}_1$ mod $q$ in the previous case, if test = true, necessarily test = false, and vice versa. Both cases being symmetric, we consider the case test = true. From $A$'s view, before outputting $c_{\mu}$ the only fixed information relative to $k_1$ is that contained $c_{k_1} = (g_{q_1}^{b_{1}}, (g_{q_1}^{r_{1}} f^{h_{1}})^{sk_1} f^{k_1})$. This fixes $\pi_0 := b_1 \cdot sk_1 + k_1$ mod $q$. However from $A$'s view, given pk_1, the random variable $sk_1$ follows a distribution $\delta_s$-close to $U(Z/qZ)$. Thus $k_1$ also follows a distribution $\delta_s$-close to $U(Z/qZ)$. Now suppose $A$ returns $c_{\mu} = (z_{\mu}, z_{\mu}^{sk_1} f^{h_{\mu}})$ where $z_{\mu} \in \hat{G}$. If test = true, then $\mu \cdot P + B_{1,j} = k_1 W_j$, and $A$ has fixed the correct value of $k_1$, this occurs with probability $\leq 1/q + \delta_s$.

(b) $\mu \neq \mu'$ mod $q$ but $\mu - \mu' = w_{j}(k_1 - \tilde{k}_1)$ mod $q$, i.e. $b_{\mu} = w_{j} b_1$ mod $q$. This results in $S$ aborting on $\mu'$ in Games if and only if $S$ aborts on $\mu$ in Games. 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:
\[
\mu = k_1 w_j - \nu_{1,j} \iff \mu' + w_j (k_1 - \hat{k}_1) = k_1 w_j - \nu_{1,j} \iff \mu' = \hat{k}_1 w_j - \nu_{1,j}.
\]

(c) $(\mu \not\equiv \mu' \mod q)$ and $(\mu - \mu' \not\equiv w_j (k_1 - \hat{k}_1) \mod q)$. We here consider three sub-cases:

i. Either $\text{test} = \text{test}' = \text{false}$; this results in identical views for $\mathcal{A}$.

ii. Either $\text{test}' = \text{true}$; this means that:
\[
\mu' = \hat{k}_1 w_j - \nu_{1,j} \mod q.
\]

Now since $\mu - \mu' \not\equiv w_j (k_1 - \hat{k}_1) \mod q$ necessarily $\text{test} = \text{false}$. 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 $P := (sk_1, \tilde{sk}_1, k_1, \hat{k}_1) \in (\mathbb{Z}/q\mathbb{Z})^4$, conditioned on $\mathcal{A}$’s view. For clarity, we first recall the expression of $c_{k_1}$ received by $\mathcal{A}$:
\[
c_{k_1} = (g_q^{r_1} f_{p_1}^{k_1}, p_{k_1}^{r_1} f_{\tilde{sk}_1} b_1 + \hat{k}_1)
\]
where $b_1 \not\equiv 0 \mod q$. We also recall the expression of $c_\mu$, sent by $\mathcal{A}$ to $\mathcal{S}$. Since $c_\mu$ decrypts to $\mu'$ with decryption key $\tilde{sk}_1$, we can write:
\[
c_\mu = (z_\mu f_{b_\mu}, z_\mu \tilde{sk}_1 f_{\mu'} + b_\mu \tilde{sk}_1).
\]
Let us denote $\pi_0 := \tilde{sk}_1 b_1 + \hat{k}_1 \mod q$ and $\pi_1 := \mu' + b_\mu \tilde{sk}_1$. For this case to occur, it must hold that $\mu' = \hat{k}_1 w_j - \nu_{1,j} \mod q$, so
\[
\pi_1 = \hat{k}_1 w_j - \nu_{1,j} + b_\mu \tilde{sk}_1 \mod q.
\]
Substituting $\tilde{sk}_1$ for $(\pi_0 - \hat{k}_1)b_1^{-1}$ yields:
\[
\pi_1 = \hat{k}_1 w_j - \nu_{1,j} + b_\mu b_1^{-1} (\pi_0 - \hat{k}_1) \mod q
\]
\[
\iff \pi_1 + \nu_{1,j} - b_\mu b_1^{-1} \pi_0 = \hat{k}_1 (w_j - b_\mu b_1^{-1}) \mod q
\]
As we dealt with $b_\mu = w_j b_1 \mod q$ in case (b), here $w_j - b_\mu b_1^{-1}$ is invertible mod $q$ so we can write:
\[
\hat{k}_1 = (\pi_1 + \nu_{1,j} - b_\mu b_1^{-1} \pi_0) (w_j - b_\mu b_1^{-1})^{-1} \mod q
\] (1)
where $\pi_0, b_1$ are fixed by $c_{k_1}$; $\pi_1, b_\mu$ are fixed by $c_\mu$; $w_j$ 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 $c_\mu$), 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 $c_\mu$), the $\delta_s$-smoothness of the CL scheme.
ensures that \( \hat{k}_1 \) follows a distribution \( \delta_x \)-close to \( U(Z/qZ) \). If the current case occurs, Eq. [1] must hold, thus from being given a view where \( \hat{k}_1 \) follows a distribution \( \delta_x \)-close to \( U(Z/qZ) \), \( \mathcal{A} \) succeeds in fixing this random variable to be the exact value used by \( S \). This occurs with probability \( \leq 1/q + \delta_x \).

iii. Else test = true; this means that \( \mu = k_1 w_j - \nu_{1,j} \mod q \). Since \( \mu - \mu' \neq w_j(k_1 - \hat{k}_1) \mod q \) necessarily test' fails, and \( \mathcal{A} \)'s view differs.

Reasoning as in the previous case, but setting \( \pi_0 := sk_1 b_1 + k_1 \mod q \) and \( \pi_1 := \mu + b_\mu sk_1 \), one demonstrates that this case occurs with probability \( \leq 1/q + \delta_x \).

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

\[
|\Pr[\mathcal{E}_3] - \Pr[\mathcal{E}_2]| \leq 3/q + 4\delta_x.
\]

**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:

\[
|\Pr[\mathcal{E}_4] - \Pr[\mathcal{E}_3]| \leq \delta_{\text{HSM}}.
\]

**Game_4 to Game_5.** In Game_5 \( S \) 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:

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

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

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

which concludes the proof of the claim.

3. As a consequence of the different values \( k \) and \( \hat{k}_1 \), there is also a difference in the values \( k_1 \cdot R \) and \( \hat{k}_1 \cdot R = P - \sum_{i \in S, i \neq 1} k_i \cdot R \) after rewinding in phase 4.

However, they follow the same distribution, and they can be distinguished if \( k_1 \) and \( \hat{k}_1 \) 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 \( c_{k_1} \) 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. \( S \) does not know \( \sigma_1 \), 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) \( s = \hat{k}(m + r x) \). So \( s_1 = \hat{k} m + r(\hat{k} x - \sum_{j \in S, j \neq 1} \sigma_j) \), and \( S \) is implicitly setting \( \hat{\sigma}_1 := \hat{k} x - \sum_{j \in S, j \neq 1} \sigma_j \) s.t. \( \hat{k} x = \hat{\sigma}_1 + \sum_{j \in S, j \neq 1} \sigma_j \).
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 $R = k^{-1}P = (k_1 + \sum_{j\in S, j\neq 1} k_j)^{-1} \cdot P$. Therefore the value $s_1$ computed by $S$ 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.

**Indistinguishability of identification procedure.** We just proved that 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 19. For an abort of type 3 identifying the culprit is trivial, and there is no impact on the view $A$ has of real and simulated executions. If an abort of type 1 or 2 occurs, in all of our considered game steps $S$ can honestly perform the proof for relation $R_{Dec}$ as would $P_1$, hence $A$’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 $S$ 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 $S$ follows the real identification procedure, hence the simulation is perfect. For type 4 the only difference is that $S$ simulates $\pi_j^{\log}$, so $A$’s view is indistinguishable.

**Concluding the proof.** The forger $S$ simulating $A$’s environment can detect whether we are in a semi-correct execution or not. Consequently $S$ always knows how to simulate $A$’s view and all simulations are indistinguishable from real executions of the protocol. Moreover if $A$, having corrupted up to $t$ parties in the threshold EC-DSA protocol, outputs a forgery, since $S$ set up with $A$ the same public key $Q$ it received from its’ EC-DSA challenger, and randomness $R$ it received from $\Theta^R$, $S$ 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 2. 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 $\delta_s$-smooth; the HSM problem is $\delta_{\text{HSM}}$-hard, and the commitment scheme is non-malleable and equivocal, 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 3. Assuming EC-DSA is e-eu-cma; the DL assumption holds in $G$; the strong root and C-low order assumptions hold for Gen; the CL encryption scheme is $\delta_s$-smooth; the HSM problem is $\delta_{\text{HSM}}$-hard, and the commitment scheme is non-malleable and equivocal, then the $(n - 1, n)$-threshold EC-DSA protocol of Fig. 3-4-5-6 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, we present details below.

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 $S$ chooses a new special player. Since there is only one special player, $S$ 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, the aforementioned rewind does not introduce the risk that $A$ 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. 8.

![Fig. 8: Key Generation protocol when $t = n - 1$](image)

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, $S$ must be capable of providing a consistent internal state whenever $A$ corrupts an honest player. We point out that using Feldman-VSS causes issues in the simulation, since the simulator – simulating the special player $P_s$ – computes a polynomial $p_s(X)$, for which it can give at most $t$ consistent shares which pass the verification check. Indeed, $t + 1$ shares define in a unique way $p_s(X)$, which has an unknown degree zero coefficient. As a result, $S$ will send $t$ consistent shares and $n - t$ inconsistent shares with overwhelming probability.

---

8 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.
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, $\mathcal{A}$ 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 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.

**Proof of Theorem 3**

**Notation.** Before proving the theorem, let us introduce some notations. The sets of the indices of all players and all corrupted players are denoted $P$ and $C$ 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 $H$ contains indices of all honest players except $P_*$, while $NC$ contains indices of all non corrupted players including $P_*$. Hence $H = P \setminus (C \cup \{P_*\})$ and $NC = P \setminus C$. The sets $H, C, NC$ are dynamically updated with new corruptions throughout the protocol. In particular, $C$ grows in size with the condition that $|C| \leq n - 1$, taking elements from $NC$. 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 $NC \leftarrow NC \setminus \{P\}$ then $H \leftarrow H \setminus \{P\}$. Finally, all values belonging to the special player $P_*$ are indexed with the symbol $\ast$.

**Simulating Key Generation.**

1. $\mathcal{S}$ receives a public key $Q$ from it’s EC-DSA challenger.
2. For $i \in NC$, $\mathcal{S}$ samples $w_i \overset{\$}{\triangleright} \mathbb{Z}/q\mathbb{Z}$ and computes $[kgc_i, kgd_i] \leftarrow \text{Com}(w_i, P)$.
3. For $i \in NC$, $\mathcal{S}$ samples CL encryption key pairs $(pk_i, sk_i) \overset{\$}{\leftarrow} \text{KeyGen}(1^\lambda)$.
4. $\mathcal{S}$ broadcasts $\{kgc_i\}_{i \in NC}$ and $\{pk_i\}_{i \in NC}$, before receiving $\{kgd_j\}_{j \in C}$ and the public keys $\{pk_j\}_{j \in C}$ from $\mathcal{A}$.
5. $\mathcal{S}$ broadcasts $\{kgd_j\}_{i \in NC}$ and receives $\{kgd_j\}_{j \in C}$. For $i \in P$, let $W_i \leftarrow \text{Open}(kgc_i, kgd_i)$ 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 $\overline{kgd}_*$ which commits to $\overline{W}_* := Q - \sum_{j \neq \ast} W_j$.
7. $\mathcal{S}$ simulates $\pi^kg$ (the ZKPoK that it knows $w_*$ corresponding to $\overline{W}_*$) and honestly performs the proofs $\pi^kg_i$ for $i \in H$. Then, for $j \in C$, $\mathcal{S}$ receives from $\mathcal{A}$ a ZKPoK of $w_j$ satisfying $W_j := w_j \cdot P$; from which $\mathcal{S}$ can extract $w_j$. 

37
Simulating Key Refresh. In the event of a normal Key Refresh (i.e. which is not due to \( \mathcal{A} \) rewinding to switch special players), \( \mathcal{A} \) simply runs the real Key Refresh sub-protocol for all players in 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 \( P^*_s \), then \( \mathcal{A} \) rewinds the adversary and chooses a new special player \( P^*_s^\text{new} \) among the honest parties \( P_i \) for \( i \in H \). 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 \( P^*_s \) has consistent values, so that \( P^*_s^\text{new} \) is the unique inconsistent (i.e. special) player. Note that throughout KRSS, we assume \( P^*_s \) is not corrupted, and \( \mathcal{A} \) can corrupt at most \( n-2 \) of the remaining \( n-1 \) players.

Without loss of generality we set \( P_1 := P^*_s \) and \( P_2 := P^*_s^\text{new} \). For \( i \in P \) we denote \( u_i \in \mathbb{Z}/q\mathbb{Z} \) 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 \( \mathcal{A} \) does not know \( u_1 \).

If a KRSS occurs, \( \mathcal{A} \) 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 \( \alpha \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z} \) and let \( \beta := v_{1,1} + v_{1,2} - \alpha \).
- Then set \( Q_{1,1} := -Q_1 + \alpha P, Q_{1,2} := Q_1 + \beta P \) and for each \( j \in P, j > 2 \) set \( Q_{1,j} := v_{1,j} P \).
- Compute \( Q_{2,j} := v_{2,j} P \) for all \( j \in P, j \neq 2 \) as in the real protocol.
- For \( j \in P \) compute ciphertexts \( C_{1,j} \leftarrow \text{Enc}(pk_j, v_{1,j}) \) and \( C_{2,j} \leftarrow \text{Enc}(pk_j, v_{2,j}) \) as per the protocol. Simulate proofs \( \pi_{1,j}^r \) and \( \pi_{1,2}^r \) but run all other proofs \( \{\pi_{1,j}^r\}_{j \in P, j > 2} \) and \( \{\pi_{2,j}^r\}_{j \in P} \) as in the real protocol.
- After having received all the \( \{Q_{i,j}\}_{i \in P, j > 2, j \in P} \), \( \mathcal{A} \) computes \( Q_{i}^\text{new} = Q_i + \sum_{j \in P} Q_{j,i} \) for each \( i \neq 2 \). It then rewinds the sub-protocol and changes \( Q_{2,2} \) to

\[
Q_{2,2} := Q - \sum_{j \neq 2} Q_{j,1}^\text{new} - Q_2 - \sum_{i \neq 2} Q_{i,1}^\text{new}
\]

With this choice of \( Q_{2,2}^\text{new} \), \( Q_2^\text{new} = Q_2 + \sum_i Q_{i,2}^\text{new} \) is such that \( Q = \sum_{i \in P} Q_i^\text{new} \).
- Erase all values \( v_{1,i} \) and \( Q_{1,i} \)

Notice that with this choice of \( Q_{1,i} \), there are no inconsistencies for \( P_1 \) and it knows the discrete log of its' point. Furthermore, thanks to the values \( \alpha \) and \( \beta \) 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 \( \Theta \) to obtain a uniformly random point \( R = (r_x, r_y) \) in \( \mathbb{G} \):

38
Phase 2: Recall that during the regular run of the protocol, $P$ simulates Pre-signing and Signing - Description of $S$.

The simulator simulates $P_i$ for each $i \in \text{NC}$ 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 $sk_i$, and $P_i$’s EC-DSA public key share $W = w_i \cdot P$; it does not know $w_i$. However the simulator knows the shares $w_i$ of all other players ($i \in P \setminus \{\ast\}$) from the Schnorr proofs in Key Generation phase (for $i \in C$) or because it computed them (for $i \in H$).

The simulation of the Pre-Signing and Signing protocols is based on $\text{CGG}^{+20}$ and $\text{CCL}^{+20}$, with adaptations considering previously defined dynamic sets of players ($\text{NC, H, C}$). For each execution all parties in $P$ participate. This implies that $\{w_i\}_{i \in [n]}$ are long term secrets. In the following simulation $S$ aborts whenever the protocol is supposed to abort, i.e., whenever $A$ 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 $S$: For all $i \in H$, i.e. honest – but not special – players $P_i$, $S$ just runs the protocol as would $P_i$ in a real execution. Hence in the following phases we only describe how $S$ simulates $P_i$.

Phase 1: $S$ samples $k_s, \gamma_s \sim \mathbb{Z}/q\mathbb{Z}$, $r_s \sim \{\hat{A}\}$ uniformly at random. It computes $c_{k_s} \leftarrow \text{Enc}(pk_s, k_s; r_s)$, the associated ZKAoK $\pi_s$, and $[c_s, d_s] \leftarrow \text{Com}(\gamma_s, P)$. It broadcasts $c_s, c_{k_s}, \pi_s$ before receiving $\{c_j, c_{k_j}, \pi_j\}_{j \in C}$ from $A$. $S$ checks the proofs are valid and extracts the encrypted values $\{k_j\}_{j \in C}$ from which it computes $k := \sum_{i \in P} k_i$.

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

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

(b) **Respondent for MtA with $k_j, j \in P \setminus \{\ast\}$ and $\gamma_s$:** $S$ runs the real sub-protocol, as it knows $\gamma_s$.

Recall that $S$ knows $k_j$ from extraction in Phase 1, it also knows its own shares $\beta_{j,i}$ for $i \in \text{NC}$, hence $S$ can compute $P_j$’s shares $\alpha_{j,i} = k_j \gamma_i - \beta_{j,i} \mod q$.

(c) **Initiator for MtAwc with $k_s$ and $w_j, j \in P \setminus \{\ast\}$:** $S$ runs the real sub-protocol, as it knows $k_i$ for $i \in \text{NC}$. Notice that $S$ chose $w_i$ for $i \in H$ as in the real protocol, while for $j \in C$, $S$ extracted $w_j$ from $\pi_j^\text{Awc}$ in KeyGen. The only unknown share of $x$ is the special player’s $w_s$. For $j \in P \setminus \{\ast\}$, $S$ runs the real sub-protocol; decrypting $c_{k,w_j}$ to obtain
Phase 3: $S$ responds for MtAwc with updates $\sigma$ broadcasts is semi-correct or not, every time a player $P$ further more, up until the moment $\mu$ since it knows all the values $T$ computes: $\delta$ does not know the internal values from the MtA and MtAwc nor can $S$ compute the individual values $w$ for each $i,j \in \mathbb{C}$; and checking that $\mu_{*,j}P + B_{*,j} = k_{*,j}W_{ij}$. If so, since $S$ also knows $k_{*,j}$ and $w_{ij}$, it computes $\nu_{*,j} = k_{*,j}w_{ij} - \mu_{*,j} \mod q$.

(d) **Respondent for MtAwc with** $k_{*,j}, j \in \mathbb{P} \setminus \{\ast\}$ and $w_\ast$: $S$ knows $W_{\ast} = w_\ast \cdot P$ but not $w_\ast$, so it samples a random $\mu_{*,j} \leftarrow \mathbb{Z}/q\mathbb{Z}$ and sets $c_{k_{*,j}} \leftarrow \text{Enc}(pk_{*,j}, \mu_{*,j})$, and $B_{*,j} := k_{*,j} - \mu_{*,j} \cdot P$. Finally $S$ sends the ciphertexts

Note that at this point $S$ knows:

- $k_{i}$ for each $i \in \mathbb{P}, w_{j}$ for each $j \in \mathbb{P} \setminus \{\ast\}$
- $\alpha_{i,j}, i \in \mathbb{NC}, j \in \mathbb{P} \setminus \{i\}$ as initiator for MtA, $\alpha_{j,i}, \beta_{j,i}, i \in \mathbb{NC}, j \in \mathbb{P} \setminus \{i\}$ as respondent for MtA
- $\mu_{i,j} \cdot \nu_{i,j}, i \in \mathbb{NC}, j \in \mathbb{P} \setminus \{i\}$ as initiator for MtAwc, $\mu_{j,i}, \nu_{j,i}, i \in \mathbb{NC}, j \in \mathbb{P} \setminus \{i\}$ as respondent for MtAwc

$S$ computes $\delta_{i}$ for the special player and $\delta_{i}$ for $i \in \mathbb{H}$ as per protocol.

Note that $S$ does not know the internal values from the MtA and MtAwc protocols executed by two players that are both controlled by the adversary. Thus $S$ is not able to compute the individual values $\sigma_{j}$ and $\delta_{j}$ for $j \in \mathbb{C}$; nor can $S$ compute $\sigma_{i}$ since it doesn’t know the value $w_{i}$. However $S$ can compute:

$$
\sigma_{C} = \sum_{i \in \mathbb{C}} \sigma_{i} = \sum_{i \in \mathbb{C}} (k_{i}w_{i} + \sum_{j \in \mathbb{P} \setminus \{i\}} \mu_{i,j} + \sum_{j \in \mathbb{P} \setminus \{i\}} \nu_{j,i})
= \sum_{i \in \mathbb{C}} \sum_{j \in \mathbb{P} \setminus \{i\}} (\mu_{i,j} + \nu_{j,i}) + \sum_{i \in \mathbb{C}} k_{i}w_{i}
= \sum_{i \in \mathbb{C}} \sum_{j \in \mathbb{C}, j \neq i} (\mu_{i,j} + \nu_{j,i}) + \sum_{i \in \mathbb{C}} \sum_{j \in \mathbb{NC}} (\mu_{i,j} + \nu_{j,i}) + \sum_{i \in \mathbb{C}} k_{i}w_{i}
= \sum_{i \in \mathbb{C}} \sum_{j \in \mathbb{C}, j \neq i} k_{i}w_{j} + \sum_{i \in \mathbb{C}} \sum_{j \in \mathbb{NC}} (\mu_{i,j} + \nu_{j,i}) + \sum_{i \in \mathbb{C}} k_{i}w_{i}
= \sum_{i \in \mathbb{C}} \sum_{j \in \mathbb{NC}} (\mu_{i,j} + \nu_{j,i}) + \sum_{i \in \mathbb{C}} \sum_{j \in \mathbb{C}} k_{i}w_{j}
$$

since it knows all the values $\{k_{i}\}_{i \in \mathbb{P}}, \{w_{j}\}_{j \in \mathbb{P}, j \neq \ast}, \mu_{i,j}$ and $\nu_{j,i}$ in MtAwc with the honest players and $\mu_{*,i}, \nu_{*,i}$ from special player.

Furthermore, up until the moment $\sigma_{C}$ is used to check whether the execution is semi-correct or not, every time a player $P_{i}$ for some $i \in \mathbb{NC}$ is corrupted, $S$ updates $\sigma_{C} \leftarrow \sigma_{C} + \sigma_{i}$. If the special player is corrupted, the simulator rewinds, and $\sigma_{C}$ is recomputed.

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

Phase 4: $\delta$ broadcasts $d_\ast$ which decommits to $\Gamma_\ast$, and for all $j \in C$, $\mathcal{A}$ reveals $d_j$ which decommit to $\Gamma_j$. $\delta$ honestly performs the ZK proof $\pi_j^\ast$; and receives $\Gamma_j^\ast$, from which $\delta$ can extract $\gamma_j$. These are consistent with the values used in Phase 1 thanks to the binding property of the commitment scheme. Now $\delta$ can compute

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

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

$$\bar{R} = \bar{\delta}^{-1} \cdot \sum_{i \in P} \Gamma_i.$$ 

Then using the values $\{k_j\}_{j \in C}$ extracted in Phase 1, and its own values $\{k_i\}_{i \in NC}$, $\delta$ checks if $\sum_{i \in P} k_i \cdot \bar{R} = P$. If equality holds then $\bar{R} = k^{-1} \cdot P$ and $\bar{\delta} = k_\gamma = \delta$.

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

$$\sum_{i \in P} k_i \bar{R} = P \quad \text{and} \quad \bar{\sigma}_C = \sigma_C.$$ 

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 $\delta$ and $\bar{\delta}$ avoids the need for proofs of affine transformation which were necessary in [CGG+20] to attain security against malicious adversaries.

Now $\delta$ adapts its behaviour depending on the type of execution:

- **Semi-correct execution:**
  
  (a) $\delta$ invokes oracle $\mathcal{O}_R$ to obtain $R = (r_x, r_y)$.
  
  (b) $\delta$ sets $\bar{\Gamma}_s := \bar{\delta} \cdot R - \sum_{i \in P, i \neq s} \Gamma_i$, so that $R = \bar{\delta}^{-1} \left( \bar{\Gamma}_s + \sum_{i \in P, i \neq s} \Gamma_i \right)$.

  Then $\delta$ rewinds $\mathcal{A}$ to the decommitment step in Phase 4, and equivocates $P_s$’s commitment so that it decommits to $\bar{\Gamma}_s$ instead of $\Gamma_s$.

- **Non semi-correct execution:** $\delta$ simply moves on to Phase 5.

Phase 5: - **Semi-correct execution:** $\delta$ publishes $\bar{R}_s = P - \sum_{i \in P \setminus \{s\}} k_i \bar{R}$ together with $\pi_s^\ast$: a simulated ZKP of consistency with $c_k = \text{Enc}(pk_s, k_s; r_s)$ (note that in this case $\bar{R}_s \neq k_s \cdot R$ due to the rewinding).
– Non semi-correct execution: $S$ publishes $\bar{R}_* := k_* \cdot R$ together with $\pi'_j$: a real ZKP of consistency with $c_k$ (this needn’t be simulated).

Phase 6: – Semi-correct execution: $S$ publishes $S_* := Q - \sum_{j \in P \setminus \{s\}} \sigma_j R$ together with $\pi''_j$: 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: $S$ publishes $S_* := \sigma_* R$ together with $\pi''_j$: a real ZKP of consistency with $T_*$ (this needn’t be simulated).

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

Phase 7: $S$ invokes the second oracle $O_{\text{Sign}}(sk, m; R)$ 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, $S$ receives the valid signature $(r, s)$ on $m$, where $r = r_x \mod q$.

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

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

Simulating Identification

Simulating Identification of aborts in Key Generation – Description of $S$. If an abort occurs in the Key Generation protocol, $S$ runs the identification protocol as would an honest $P_i$ for each $i \in \text{NC}$ (i.e. as described on page 15. Furthermore, if some player $P$ raises a compliant against $P_*$ (simulated by $S$), 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 $S$. For all $i \in H$, i.e. honest – but not special – players $P_i$, $S$ just runs the identification procedure as would $P_i$ in a real execution. Hence in the following phases we only describe how $S$ simulates $P_i$. Consider the problematic types of abort listed on
For an abort of type 1, 3 or 5 occurs, $S$ runs the real identification procedure.

If an abort of type 2 occurs due to $S$ announcing that the check on $\mu_{j,*}$ fails (for some $j \in C$), it runs the real identification procedure. Conversely, if some player $P_j$ for $j \in C$ 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 $R_{\text{Dec}}$ provided by $P_j$ is valid), then since the point $B_{j,*}$ sent by $S$ to $P_j$ was computed as $B_{j,*} := k_j \cdot W_s - \mu_{j,*}$, 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 $S$ knows $\nu_{j,*}$ such that $B_{j,*} = \nu_{j,*} \cdot P_s$, and the simulation remains undetected.

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

**Indistinguishability of real and simulated executions against adaptive adversaries**

**The simulation of a semi-correct execution**

**Lemma 4.** Assuming the strong root and C-low order assumptions hold for $\text{Gen}$; the CL encryption scheme is $\delta_\cdot$-smooth; the HSM problem is $\delta_{\text{HSM}}$-hard; 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 4 very much resembles that of Lemma 2. Hence many details are here omitted.

**Proof.** Since, in all considered protocols, $S$ simulates parties $P_i$ for $i \in H$ by running the real protocol exactly as would $P_i$, one only needs to prove that $S$’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.

**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, $S$ 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 \( \{c_{k_jw_*}\}_{j \in \text{P} \setminus \{\ast\}} \) as in a real execution of the protocol. However as in the static case (replacing '1' with '*' ), \( \mathcal{S} \) can extract \( k_j \) from \( \pi_j \) for each \( j \in \text{C} \) and it knows \( k_j \) for each \( j \in \text{H} \). 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 \( b_* = \delta \cdot R - \sum_{i \neq \ast} \Gamma_i, \) and equivocates its commitment \( c_* \) s.t. \( d_* \) decommits to \( b_* \). 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 \( \mathcal{S} \) with all players \( \text{P} \)). And using the same reasoning as in proof of Lemma 2, one can demonstrate that the following claim holds:

**Claim.** If the CL encryption scheme is \( \delta_s \)-smooth and the HSM problem is \( \delta_{\text{HSM}} \)-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 \( k \) with probability greater than \( 2\delta_{\text{HSM}} + 3/q + 4\delta_s \).

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 \( \bar{k} \in \mathbb{Z}/q\mathbb{Z} \) the randomness (unknown to \( \mathcal{S} \)) used by oracle \( \mathcal{O}_R \) to produce \( R \). With overwhelming probability, \( k \neq \bar{k} \). Hence there is also a difference in the values \( k_* \cdot R \) and \( \bar{k} \cdot R = P - \sum_{i \neq \ast} k_i \cdot R \) 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 \( \pi_* \)), 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 \( S_* = Q - \sum_{i \neq \ast} S_i \).

5. \( \mathcal{S} \) does not know \( \sigma_* \), 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 5 is essentially identical to that in the static case, substituting \( P_1 \) for \( P_* \).

**Lemma 5.** Assuming the strong root and C-low order assumptions hold for \( \text{Gen} \); 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 2 and 5 it holds that if the strong root an C-low order assumptions hold for \( \text{Gen} \); the CL encryption scheme is ind-CPA-secure and the commitment scheme is non malleable and equivocable, then the \((n - 1, n)\) ECDSA protocol described in Figures 8, 4, 5 and 6 is enhanced threshold existentially unforgeable against adaptive adversaries.
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. 9. For our choice on the size of the discriminant $\Delta_K$ 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. 9 clearly demonstrates the impressive efficiency gains we attain, reducing by a factor 10 the bandwidth consumption compared to [CGG+20].

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Curve size</th>
<th>$\lambda$ (bits)</th>
<th>$\Delta_K$ (bits)</th>
<th>$N$ (bits)</th>
<th>Total Signing (KBytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canetti et al.’s 6 rounds</td>
<td>256</td>
<td>112</td>
<td>-</td>
<td>2048</td>
<td>31.3$t + 1.0$</td>
</tr>
<tr>
<td>Canetti et al.’s 3 rounds</td>
<td>256</td>
<td>112</td>
<td>-</td>
<td>2048</td>
<td>31.6$t + 1.3$</td>
</tr>
<tr>
<td>Ours</td>
<td>256</td>
<td>112</td>
<td>1348</td>
<td>-</td>
<td>$3.4t + 2.0$</td>
</tr>
<tr>
<td>Canetti et al.’s 6 rounds</td>
<td>256</td>
<td>128</td>
<td>-</td>
<td>3072</td>
<td>45.1$t + 1.5$</td>
</tr>
<tr>
<td>Canetti et al.’s 3 rounds</td>
<td>256</td>
<td>128</td>
<td>-</td>
<td>3072</td>
<td>45.3$t + 1.8$</td>
</tr>
<tr>
<td>Ours</td>
<td>256</td>
<td>128</td>
<td>1827</td>
<td>-</td>
<td>$4.1t + 2.3$</td>
</tr>
</tbody>
</table>

Fig. 9: 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 420KB of data being transmitted in their protocol, as opposed to 28KB in ours; i.e. a reduction by a factor 15.

**Acknowledgements.** The work of the first and third authors was supported by the French ANR SANGRIA project (ANR-21-CE39-0006). The work of the second author was in part support by the Programma ricerca di ateneo UNICT 2020-22 linea 2. The work of the third author was supported by the French PEPR Cybersecurit´e SecureCompute project (ANR-22-PECY-0003). The work of fifth author was supported by ERC (ERC-2020-COG).

**References**


