UC Non-Interactive, Proactive, Threshold ECDSA

Ran Canetti* Nikolaos Makriyannis† Udi Peled†

April 28, 2020

Abstract

Building on the protocol of Gennaro and Goldfeder (CCS '18), we present a threshold ECDSA protocol, for any number of signatories and any threshold, that improves as follows over the state of the art:

- Signature generation takes only 4 rounds (down from the current 8 rounds), with a comparable computational cost. Furthermore, 3 of these rounds can take place in a preprocessing stage before the signed message is known, lending to the first *non-interactive* threshold ECDSA protocol.
- The protocol withstands adaptive corruption of signatories. Furthermore, it includes a periodic refresh mechanism and offers full proactive security.
- The protocol realizes an ideal threshold signature functionality within the UC framework, in the global random oracle model, assuming Strong RSA, semantic security of Paillier encryption, and a somewhat enhanced variant of existential unforgeability of ECDSA.

These properties (low latency, compatibility with cold-wallet architectures, proactive security, and composable security) make the protocol ideal for threshold wallets for ECDSA-based cryptocurrencies.

*Boston University. Email: canetti@bu.edu.
†Fireblocks. Emails: nikos@fireblocks.com, udi@fireblocks.com

Contents

1 Introduction

Introduced by Desmedt [18] and Desmedt and Frankel [19], threshold signatures allow a number of signatories to share the capability to digitally sign messages, so that a given message is signed if and only if a certain threshold of the signatories agree to sign it. In more detail, a t-out-of-n threshold signature scheme is a mechanism whereby a set of n signatories, presented with a message m, jointly and interactively compute a *signature* σ such that (1) if at least t of the signatories agree to sign m, then the pair m, σ is accepted as a valid by a pre-determined public verification algorithm, and (2) no attacker that controls up to $t-1$ signatories can forge signatures – namely, it cannot come up with a pair m', σ' such that the verification algorithm accepts σ' as a valid signature on m', if the latter was never signed before.

Threshold signatures are an instance of "threshold cryptography" which, in turn, is one of the main application areas of the more general paradigm of secure multi-party computation. Threshold cryptography offers an additional layer of security to a cryptographic task (compared to the standard underlying task, e.g. signatures), by distributing the sensitive capabilities (such as signing or decrypting) among multiple servers/devices, thus realizing the "no single point of failure" paradigm. The literature is rich with many threshold cryptography schemes: El-Gamal, RSA, Schnorr, Cramer-Shoup, ECEIS, to name a few [38, 17, 37, 39, 12].

With the advent of blockchain technologies and cryptocurencies in the past decade, there has been a strong renewed interest in threshold cryptography and threshold signatures in particular. Specifically, because transactions are made possible via digital signatures, many stakeholders are looking to perform signature-generation in a distributed way, and many companies are now offering solutions based on (or in combination with) threshold cryptography.[1]

Threshold ECDSA. The *digital signature algorithm* (DSA) [31] in its elliptic curve variant (ECDSA) [35] is one of the most widely used signature schemes. ECDSA has received a lot of attention from the cryptography community because, apart from its popularity, it is viewed as somewhat "threshold-unfriendly", i.e. (naive) threshold protocols for ECDSA require heavy cryptographic machinery over many communication rounds. Early attempts towards making threshold (EC)DSA practically efficient include Gennaro et al. [25] in the honest majority setting and MacKenzie and Reiter [34] in the two-party setting.

In recent years, there has been an abundance of protocols for threshold ECDSA [26, 2, 24, 32, 33, 20, 21, 15, 16] that support any number n of parties and allow any threshold $t < n$. The protocols that stand out here are the ones by Gennaro and Goldfeder [24], Lindell et al. [33] and Doerner et al. [21], and the recent work of Castagnos et al. [16].

We note that all recent protocols achieve competitive practical performance (with trade-offs between computation and communication costs depending on the tools used). Furthermore, all recent protocols require at least eight communication rounds which for many modern communication settings (involving geographically dispersed servers and/or mobile devices) is the most time-consuming resource.

1.1 Our Results.

We present a new threshold ECDSA protocol. The protocol builds on the techniques of Gennaro and Goldfeder [24] and Lindell et al. [33], but provides new functionality, and has improved efficiency and security guarantees. We first discuss the new characteristics of the protocol at high level, and then provide more details on the construction and the analysis. Figure 1 provides a rough comparison between the main cost and security guarantees of our scheme and those of Gennaro and Goldfeder [24], Lindell et al. [33], Doerner et al. [21], and Castagnos et al. [16].

Non-Interactive Signing. As seen in Figure 1, in all of these protocols the signing process is highly interactive, i.e. the parties exchange information in a sequence of rounds to compute a signature for a given message. However, in many real-life situations it is desirable to have non-interactive signature generation, namely have each signatory, having seen the message, generate is own "signature share" without having to interact with any other signatory, and then have a public algorithm for combining the signature shares into a single signature. For instance, such a mechanism is mandatory if one wants to use a "cold wallet" mechanism for some of the signatories — which is a common practice in the digital currency realm for securing non-threshold

[1] See https://www.mpcalliance.org/ for companies in the threshold cryptography space.

Signing Protocol	*Rounds*	*Group Ops*	*Ring Ops*	*Communication*	*Proactive*
Gennaro and Goldfeder [24]	9	$10n$	$20n$	$10\kappa + 20N$ (7 KiB)	✗
Lindell et al. [33] (Paillier)†	8	$80n$	$20n$	$50\kappa + 20N$ (7.5 KiB)	✗
Lindell et al. [33] (OT)†	8	$80n$	0	50κ (190 KiB)	✗
Doerner et al. [21]	$\log(n) + 6$	5	0	$10 \cdot \kappa^2$ (90 KiB)	✗
Castagnos et al. [16]*	8	$15n$	0	$100 \cdot \kappa$ (4.5 KiB)	✗
This Work: *Interactive*	4	$10n$	$90n$	$10\kappa + 50N$ (15 KiB)	✓
This Work: *Non-Int. Pre-Sign*	3	$10n$	$90n$	$10\kappa + 50N$ (15 KiB)	✓
This Work: *Non-Int. Sign*	1	0	0	κ (256 bits)	✓

Figure 1: Comparison of our scheme with those of [24, 33, 21, 16] for signing. Costs are displayed per party for an n-party protocol secure against $n-1$ corrupted parties, for computational security of 128 bits and statistical security of 80 bits. The communication column describes the number of group elements (encoded by κ bits) and ring elements (encoded by N bits) sent between each pair of parties; in parentheses we provide estimates, including the constant overhead, for concrete implementation with the curve size for Bitcoin and the standard security recommendation for Paillier, i.e. $\kappa = 256$ and $N = 2048$. (†Estimates for [33] include optimizations that do not preserve UC – c.f. Section 1.3 for further details. *We note that [16] relies on somewhat incomparable hardness assumptions. Also the protocol involves operations in a different group than the underlying elliptic curve.)

wallets. Indeed, a number of popular signature schemes do admit threshold protocols with non-interactive signing (e.g. RSA [30], BLS [1]). However, no non-interactive threshold signature for ECDSA is known.

In our protocol, the signing process can be split into two phases: A first, preprocessing, phase that takes 3 rounds and can be performed before the message is known, followed by a non-interactive step where each signatory generates its own signature share, after the message to be signed becomes known. To the best of our knowledge, this is the first threshold ECDSA in the literature that has manageable performance and allows for non-interactive signing with preprocessing. Furthermore, the non-interactive step is very efficient: it boils down to computing and sending a single field-element (i.e. 256 bits for the Bitcoin curve).

Round-Minimal Interactive Signing. We stress that, even in its interactive variant, our protocol is the most round-efficient among the state-of-the-art protocols, and thus our protocol may notably improve the performance of many applications that require ECDSA (e.g. cryptocurrency custody).

Proactive Key Refresh [36, 29, 13]. While threshold signatures do provide a significant security improvement over plain signature schemes, they may still be vulnerable to attacks that compromise all shareholders one by one, in an adaptive way, over time. This vulnerability is particularly bothersome in schemes that need to function and remain secure over long periods of time. Proactive security is designed to alleviate this concern: In a proactive threshold signature scheme, time is divided into *epochs*, such that at the end of each epoch the parties engage in a protocol for refreshing their keys and local states. The security guarantee is that the scheme remains unforgeable as long as at most $t-1$ signatories are compromised *within a single epoch,* or more precisely *in any time period that starts at the beginning of one key refreshment and ends at the end of the next key refreshment.* It is stressed that the public signature verification algorithm (and key) of the scheme remains the same throughout.

Our protocol offers a two-round key refresh phase. The refreshment is the most expensive component of our protocol: For standard choice of security parameters, computation requires roughly $400 + 330n + n^2$ RSA ring exponentiations (see Appendix C for more details). However, this may be manageable, given that the refresh is done only periodically, and it can be scheduled at times of low use of the system.

We stress that none of the other protocols in Figure 1 support proactive key refreshing. In fact, these protocols are not even known to provide traditional threshold security against an adversary that corrupts parties adaptively as the system progresses.

Security & Composability. We provide security analysis of our protocols within the Universally Composable (UC) Security framework [10]. For this purpose, we first formulate an ideal threshold signature functionality which guarantees that legitimate signatures are verifiable by the standard ECDSA verification algorithm, and, at the same time, guarantees ideal and unconditional unforgeability. We show that our protocols UC-realize this ideal functionality even in the presence of an attacker that adaptively corrupts and controls parties under the sole restriction that at most t parties are corrupted in between two consecutive refresh phases. This way, we can use universal composability to assert that the protocol remains unforgeable even when put together with arbitrary other protocols. Such a strong property is of particular importance in decentralized, complext and highly security sensitive distributed systems such as cryptocurrencies.

Security of the interactive protocol is proven assuming the unforgeability of ECDSA, the semantic security of Paillier encryption and strong-RSA. It might appear a bit unsatisfying to have the unforgeability of ECDSA as an underlying assumption, given that it is an interactive – and by no means "simple" – assumption. We do this since this is the weakest assumption that one can hope for: indeed, recall that unforgeability of ECDSA is not known to follow from any standard hardness assumption on elliptic curve groups (we do however know that ECDSA is existentially unforgeable in the generic group model [6]).

Security of the non-interactive protocol is proven under the same assumptions, but with a somewhat stronger unforgeability property of ECDSA, that considers situations where the adversary obtains, ahead of time, some "leakage" information on the random string that the signer will be using for generating the upcoming several signatures. Still, the adversary should not be able to forge signatures, even given this leakage. We call this property *enhanced unforgeability,* and demonstrate that (a) ECDSA is enhanced unforgeable in the generic group model, and (b) in some cases, enhanced ungorgeability of ECDSA follows from standard unforgeability of ECDSA, in the random oracle model.

1.2 Our Techniques

Hereafter, $\mathbb{F}_q$ denotes the finite field with q elements and $\mathcal{H} : \boldsymbol{M} \to \mathbb{F}_q$ denotes a hash function used for embedding messages into the field with q elements. Furthermore, let $(\mathbb{G}, q, g)$ denote the group-order-generator tuple associated with the ECDSA curve. We use multiplicative notation for the group-operation.

1.2.1 Background

Plain (Non-Threshold) ECDSA. For message msg with $m = \mathcal{H}(\text{msg})$ and secret key $x \in \mathbb{F}_q$, recall that ECDSA has the form $(\rho, k \cdot (m + \rho x)) \in \mathbb{F}_q^2$, where ρ is the x-projection (mod q) of the point $g^{k^{-1}} \in \mathbb{G}$, and k is a uniformly random element of $\mathbb{F}_q$. On the other hand, given public key $X = g^x \in \mathbb{G}$ and signature (ρ, σ) for msg $\in \boldsymbol{M}$, the verification algorithm accepts if ρ is the x-projection of $g^{m\sigma^{-1}} \cdot X^{\rho\sigma^{-1}}$, where $m = \mathcal{H}(\text{msg})$.

Overview of the threshold ECDSA of Gennaro and Goldfeder [24]. We first describe the basic protocol for the honest-but-curious case with security threshold $t = n - 1$, i.e. the case where all signatories follow the protocol. Each signatory (henceforth, party) $\mathcal{P}_i$ chooses a random $x_i \in \mathbb{F}_q$ and sends $X_i = g^{x_i}$ to all other parties. The public key is defined as $X = X_1 \cdot \ldots \cdot X_n \in \mathbb{G}$.[2] The secret key then corresponds to the value $x = x_1 + \ldots + x_n$ (it is stressed that no one knows x). In addition, each party $\mathcal{P}_i$ is associated with parameters for an additively homomorphic public encryption scheme (specifically, Paillier encryption). That is, all parties know $\mathcal{P}_i$'s public encryption key, and $\mathcal{P}_i$ knows its own decryption key. We write enc_i, dec_i for the encryption and decryption algorithm associated with $\mathcal{P}_i$. It is stressed that all parties can run the encryption algorithm.

To sign a message msg, the parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ generate local shares $k_1, \ldots, k_n$, respectively, of the random value $k = k_1 + \ldots + k_n$, as well as local shares $\gamma_1, \ldots, \gamma_n$, respectively, of an ephemeral value $\gamma = \gamma_1 + \ldots + \gamma_n$ which will be used to mask k. Using their respective encryption schemes, each pair of parties $\mathcal{P}_i$, $\mathcal{P}_j$ computes additive shares $\alpha_{i,j}, \hat{\alpha}_{i,j}$ for $\mathcal{P}_i$ and $\beta_{j,i}, \hat{\beta}_{j,i}$ for $\mathcal{P}_j$, such that $\alpha_{i,j} + \beta_{j,i} = \gamma_j k_i$ and $\hat{\alpha}_{i,j} + \hat{\beta}_{j,i} = x_j k_i$. In more detail, the share computation phase between $\mathcal{P}_i$ and $\mathcal{P}_j$ for computing $\alpha_{i,j}$ and $\beta_{j,i}$ proceeds as follows ($\hat{\alpha}_{i,j}$ and $\hat{\beta}_{j,i}$ are analogously constructed). Party $\mathcal{P}_i$ sends $K_i = \mathsf{enc}_i(k_i)$ to $\mathcal{P}_j$, i.e. K_i an encryption of k_i under his own public key. Then, $\mathcal{P}_j$ samples a random $\beta_{j,i}$ from a suitable range, and, using the homomorphic properties

[2] For simplicity we use additive n-out-of-n secret-sharing of the private key, instead of n-out-of-n Shamir secret-sharing that is prescribed in [24].

of the encryption scheme, $\mathcal{P}_j$ computes $D_{i,j} = (\gamma_j \odot K_i) \oplus \mathsf{enc}_i(-\beta_{j,i})$,[3] i.e. $D_{i,j}$ is an encryption of $\gamma_j k_i - \beta_{j,i}$ under $\mathcal{P}_i$'s public key. Finally, $\mathcal{P}_j$ sends $D_{i,j}$ to $\mathcal{P}_i$ who sets $\alpha_{i,j} = \mathsf{dec}_i(D_{i,j})$, and the share-computation phase terminates. Upon completion, each party $\mathcal{P}_i$ can compute $\delta_i = \gamma_i k_i + \sum_{j \neq i} \alpha_{i,j} + \beta_{i,j}$, where $\delta_1, \ldots, \delta_n$ is an additive sharing of γk, ie. $\gamma k = \delta_1 + \ldots + \delta_n$.

Next, each $\mathcal{P}_i$ sends (g^{γ_i}, δ_i) to all, and the parties compute $g^{k^{-1}} = (\prod_i g^{\gamma_i})^{(\sum_j \delta_j)^{-1}}$, and obtain their respective shares $\sigma_1, \ldots, \sigma_n$ of $\sigma = k(m + \rho x)$, by setting $\sigma_i = k_i m + \rho(x_i k_i + \sum_{j \neq i} \hat{\alpha}_{i,j} + \hat{\beta}_{i,j})$, where $m = \mathcal{H}(\mathrm{msg})$ is the hash-value of msg and ρ is the x-projection of $g^{k^{-1}}$. Finally, each $\mathcal{P}_i$ sends σ_i to all, and the signature is set to (ρ, σ). To sum up, the protocol proceeds as follows from party $\mathcal{P}_i$'s perspective, where each item denotes a round:

1. Sample k_i, γ_i and send $K_i = \mathsf{enc}_i(k_i)$ to all.
2. When obtaining $\{K_j\}_{j \neq i}$, set $\{D_{j,i}, \hat{D}_{j,i}\}_{j \neq i}$ as prescribed, and send $(D_{j,i}, \hat{D}_{j,i})$ to $\mathcal{P}_j$, for each $j \neq i$.
3. When obtaining $\{(D_{i,j}, \hat{D}_{i,j})\}_{j \neq i}$, set δ_i as prescribed, and send $(\Gamma_i = g^{\gamma_i}, \delta_i)$ to all.
4. When obtaining $\{(\Gamma_j, \delta_j)\}_{j \neq i}$, set σ_i as prescribed, and send it to all.

Output. When obtaining $\{\sigma_j\}_{j \neq i}$, set σ and ρ as prescribed, and output (ρ, σ).

The above protocol takes four rounds of communication. For security, it can be seen that, *if everything was computed correctly*, then up to the point where the σ_i's are released, no coalition of up to $n-1$ parties gains any information on the secret key x. Furthermore, releasing σ_i is equivalent to releasing the signature (ρ, σ).

However, if a corrupted party deviates from the specification of the protocol, then releasing an honest party's (maliciously influenced) signature-share σ_i may reveal information about the secret key share (potentially the entirety of it). To mitigate this problem, Gennaro and Goldfeder [24] devise a special-purpose, clever technique that allows the parties to verify the validity of the signature-shares before releasing them. However, this alternative technique ends up adding five rounds of communication.

1.2.2 Our Approach

Using the above blueprint, we show how the parties can verify the validity of the signature shares without adding any rounds on top of the 4 rounds of the basic protocol, and at a comparable computational cost to that of [24]. Interestingly, we achieve this result by employing the "generic" (and often deemed prohibitively expensive) GMW-approach of proving in zero-knowledge the validity of every computation along the way, with optimizations owing to the nature of the signature functionality. Furthermore, our approach preserves the natural property of the basic protocol, whereby the message is used only in the fourth and last round. This, in turn, leads to our non-interactive variant. Proactive key-refresh phases are also built in a natural way, on top of the basic protocol, with appropriate zero-knowledge proofs.

For the analysis, we take a different approach than that taken by either [24] or [33]. Recall that Gennaro and Goldfeder [24] only demonstrates that an adversary which interacts with a stand-alone instance of their protocol and (non-adaptively) corrupts $t < n$ parties cannot forge ECDSA signatures under the public key chosen by the parties. On the other hand, Lindell et al. [33] show that their protocol UC-realizes the ECDSA functionality, in the presence of an adversary that non-adaptively corrupts $t < n$ parties. The latter is indeed a stronger property than stand-alone unforgeability, in two ways: First , this result holds even when the threshold signature protocol is part of a larger system. Second, secure evaluation of the ECDSA functionality is significantly stronger than mere unforgeability. While the first strengthening is clearly needed, the second is perhaps overly strong (for instance, it implies that the distribution of the secret randomness k is almost uniform for all signatures, regardless of the message).

We take a mid-way approach: We formulate a threshold variant of the ideal signature functionality $\mathcal{F}_{\mathsf{sign}}$ of [11] and show that our protocol UC-realizes this functionality. This way, we obtain a result that holds even when our threshold signature protocol is part of a larger system. On the other hand, we avoid the need to show that our protocol UC-realizes the ECDSA functionality. This seemingly small difference turns out to be crucial: For one, this is what allows us to prove security under *adaptive* (and even mobile [36]) corruption of parties. It also allows for a number of significant simplifications in the protocol.

[3] We emphasize that $\oplus$ and $\odot$ denote homomorphic evaluation of addition and (scalar) multiplication, respectively, rather than standard addition and multiplication.

1.2.3 Protocol overview

We proceed with an overview of our protocol. For simplicity, we have omitted many of the details, especially regarding the zero-knowledge proofs, and we refer the reader to the subsequent technical sections for further details. Let $\boldsymbol{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ denote the set of parties. Let $(\mathsf{enc}_i, \mathsf{dec}_i)$ denote the Paillier encryption-decryption algorithms associated with party $\mathcal{P}_i$; the public key is specified below. Throughout, when we say that some party broadcasts a message, we mean that the party simply sends the message to all other parties.

Key-Generation. As in the basic protcol, Each $\mathcal{P}_i$ samples a local random secret share $x_i \leftarrow \mathbb{F}_q$ of the (master) secret key $x = \sum_i x_i$ and then reveals $X_i = g^{x_i}$ by committing and then decommitting to the group-element in a sequential fashion. In addition, each party $\mathcal{P}_i$ broadcasts a Schnorr NIZK (non-interactive zero-knowledge proof of knowledge) of x_i.

Auxiliary Info & Key-Refresh. Each $\mathcal{P}_i$ locally generates a Paillier key N_i and sends it to the other parties together with a NIZK that N_i is well constructed (i.e., that it is a product of suitable primes). Next, each $\mathcal{P}_i$ chooses a random secret sharing of $0 = \sum_j x_i^j$ and computes $X_i^j = g^{x_i^j}$ and $C_i^j = \mathsf{enc}_j(x_i^j)$, for every j, *including himself.*[4] $\mathcal{P}_i$ then broadcasts $(X_i^j, C_i^j)_j$, together with a NIZK that the plaintext value of C_i^j modulo q is equal to the exponent of X_i^j. The parties update their key shares by setting $x_i^* = x_i + \sum_j \mathsf{dec}_i(C_j^i) \mod q$ if all the proofs are valid *and* $\prod_k X_j^k = \mathsf{id}_{\mathbb{G}}$, for every j.

Pre-Signing. One technical innovation that differentiates our protocol from [24] is our use of the Paillier cryptosystem as a commitment scheme. Namely, the process of encrypting values under the parties' own public keys yields a commitment scheme that is perfectly binding and computationally hiding (as long as Paillier is semantically secure). Therefore, in the protocol we instruct each party to commit to γ_i and k_i by encrypting those values *under their own keys* and broadcast $G_i = \mathsf{enc}_i(\gamma_i)$ and $K_i = \mathsf{enc}_i(k_i)$.[5] Concurrently, the parties initiate the share-computation phase (for $x_j k_i = \alpha_{i,j} + \beta_{j,i}$ and $\gamma_j k_i = \hat{\alpha}_{i,j} + \hat{\beta}_{j,i}$), while proving in zero-knowledge that the values used in the multiplication are the same as the values encrypted in G_i, K_i, as well as the exponent of the public key-share $X_i = g^{x_i}$.

At the end of the presigning phase, each $\mathcal{P}_i$ has a share k_i of k (i.e. $\sum_i k_i = k$) and χ_i of kx (i.e. $\sum_i \chi_i = kx$), as well as a point $R = g^{k^{-1}} \in \mathbb{G}$ on the curve which corresponds to the nonce of the (future) signature.

The advantage of using the Paillier cryptosystem as a commitment scheme is twofold. On one hand, Paillier ciphertexts are amenable to Schnorr-type proofs for proving the correctness of a prescribed computation. On the other hand, in the security analysis, it allows the simulator to extract the adversary's secrets, because the corrupted parties' Paillier keys are extracted during the preceding auxiliary information phase. We expand on this point in the following subsection.

The main purpose of the ZK-proofs is to bypass the security pitfalls (also highlighted in [24] and [33]) that arise from using Paillier encryption (which resides in a ring of integers modulo an RSA modulus) to derive group elements on the elliptic curve associated with ECDSA. In more detail, malicious choices of k's and γ's may allow the adversary to probe bits of the honest parties' secrets which may have devastating effect. To remedy this, similarly to [24, 33], we use ZK-range proofs with purpose of "forcing" the adversary to choose values from a suitable range, thus preventing the aforementioned attack. Our own range proofs in the present paper are modified to take into consideration the "Paillier commitments" of each party.

In summary, by virtue of the "Paillier commitments" and the accompanying ZK-proofs, party $\mathcal{P}_i$ is confident that the tuple (R, k_i, χ_i) is well-formed at the end of pre-signing phase, and there is no need for additional communication rounds to verify the correctness of the tuple, as opposed to [24, 33, 21, 16].

Signing. Once a message msg is known, to generate a signature for pre-signing data (R, k_i, χ_i), each $\mathcal{P}_i$ sets $m = \mathcal{H}(\text{msg})$, computes $\rho = R|_{x\text{-axis}}$, and sends $\sigma_i = k_i m + \rho\chi_i \mod q$ to all parties. After receiving the other parties' sigmas, the parties output the signature $(\rho, \sum_i \sigma_i) = (\rho, k(m + \rho x))$.

[4]This instruction may appear rather superfluous, but it is important to our security analysis; it allows extraction of the adversary's randomness.

[5]Notice that the ciphertexts are computationally hiding and thus the adversary cannot correlate his own k's and γ's with the honest parties' values.

1.2.4 Online vs Non-Interactive Signing

Online Signing. For interactive (online signing), the parties simply run the pre-signing stage followed by the signing stage, for a total of 4 rounds.

Non-interactive Signing. To be able to sign non-interactively, the parties need to prepare some number of pre-signatures in an offline stage. That is, for some pre-signing parameter $L \in \mathbb{N}$, the parties run the pre-signing phase L-times concurrently and obtain pre-signing data $\{(\ell, R_\ell, k_i^\ell, \chi_i^\ell)\}_{\ell=1,\ldots,L}$. Later, for each signature request using pre-signing data $(\ell, R_\ell, k_i^\ell, \chi_i^\ell)$ and message msg, the parties run the signing phase for the relevant input to generate a signature. The parties then erase the pre-signing tuple $(\ell, \ldots)$. It is important to make sure that, as part of the refresh stage, any unused pre-signatures are discarded.[6]

Remark 1.1. It is stressed that the security analysis of the non-iteractive protocol is different than the online protocol, because the signature nonces (the R's) are known well in advance of the corresponding messages to be signed. As mentioned earlier, to prove security we rely on a stronger assumption about the unforgeability of the underlying (non-threshold) scheme, and we present it in more detail in the next section.

1.2.5 Security

The present section assumes some familiarity with the ideal-vs-real paradigm and the UC-framework.

Real-vs-Ideal Paradigm & UC. We prove security via the real-vs-ideal paradigm and the Universal Composability framework. Namely, we show that our protocol emulates an ideal process involving an idealized version of the task at hand, and we prove that for every adversary attacking the protocol, there is an ideal adversary (referred to as a simulator) that achieves the same goals. In the non-UC (standalone) framework, this is done using the adversary's code and by extracting the adversary's secrets (typically via rewinding).

The UC-framework augments the above paradigm with an entity, called the *environment*, that interacts with the adversary (in the real world) or the simulator (in the ideal world), together with the parties in the computation. The goal of the environment is to guess which process (real or ideal) is executed. If no environment can tell the difference between the real and ideal processes, it follows that the protocol is secure even "in the wild"; i.e. even when it is composed arbitrarily with other (cryptographic or non-cryptographic) components of some larger system.

One major technical difference between standalone-secure and UC-secure protocols is that, in the security analysis of the latter, the simulator's arsenal of extraction techniques lacks rewinding. This typically makes the protocol more complicated because it requires tools that are amenable to so-called online extraction (see e.g. the non-rewinding version of Schnorr's NIZK proof of knowledge in Fischlin [22]). Disallowing the rewinding technique in the security analysis is also one of the major obstacles towards achieving security against adaptive party corruptions.

UC-Secure Threshold ECDSA vs Threshold Signature. One important difference between our security proof and the simulation-based security proof of [33, 21] is that our protocol UC-realizes a "generic" ideal threshold signature functionality, rather than the ECDSA functionality per se. We opted for the former for the following reasons. First, it captures more accurately the purpose of our protocol; our goal is to compute unforgeable signatures that are verifiable with the ECDSA algorithm, rather than realizing the ECDSA functionality itself. Second, and more importantly, it allows us to reintroduce the rewinding technique in the security analysis, which greatly simplifies both the protocol and the security analysis, as we explain next.

Threshold Signature Ideal Functionality & UC-simulation. We define an ideal threshold signature functionality modeled after the (non-threshold) signature functionality of Canetti [11]. The definition of the functionality aims at capturing the essence of any threshold signature scheme. Namely (and very loosely):

1. Authorized sets of parties may generate valid signatures for any given message.
2. Unauthorized sets of parties cannot compute valid signatures for messages that were never signed before.

[6] Alternatively, it is possible to keep the presigning data as long as it is appropriately refreshed, i.e. by re-randomizing the pair (k_i, χ_i).

We stress that the ideal functionality is utterly oblivious to the format of the signature scheme (there are effectively no private/public keys). Consequently, when simulating the protocol in the ideal world, it suffices to simply sample random secrets for the honest parties and follow the instructions of the protocol, and the simulation is *perfect*.

To conclude the proof, it remains to show that the requirements of the ideal functionality are met (Items 1 and 2 above). In particular, the hard part is showing that the environment *cannot forge* signatures for some message that was never signed before; this is the crux of our security proof. Before we describe the proof, we stress that because it is not part of the UC-simulation per se, the unforgeability proof is allowed to "take the environment offline" and employ the entire arsenal of extraction techniques, including rewinding. What's more, this approach gives full power to the proof over the random oracle, so that any reduction may suitably program the environment's queries to the random oracle, as long as these were never queried before.

Unforgeability Proof. We show unforgeability via reduction to the unforgeability of non-threshold ECDSA. In more detail, we consider the following experiments involving a simulator attempting to simulate the environment's interaction with the honest parties.

1. In the first experiment, the simulator follows the specifications of the protocol except that:
   (a) The simulator samples an ECDSA key-pair (x, X) and fixes the public key of the threshold protocol to X (this is achieved by rewinding the environment).
   (b) The simulator extracts the corrupted parties' Paillier keys (this is achieved by programming the random oracle).
   (c) The simulator never decrypts the ciphertexts encrypted under the honest parties' Paillier keys. Rather, to carry on the simulation, the simulator extracts the relevant values from the corrupted parties' messages, using the Paillier keys extracted in Item 1b.
   (d) To compute the honest parties' ZK-proofs, the simulator invokes the zero-knowledge simulator to generate valid proofs, and programs the random oracle accordingly.
2. The second experiment is identical to the first one except that:
   (a) At the beginning of the experiment, the simulator picks a random honest party that is henceforth deemed as special (in fact, to handle adaptive corruptions, this random party is chosen afresh every time the key-refresh phase is executed. If the environment decides to corrupt the special party, then the experiment is reset to the last preceding key-refresh; by rewinding the environment).
   (b) Every time an honest party is instructed to encrypt a value under the special party's Paillier key and send it to the corrupted parties, the simulator sends a random encryption of 0 instead.
3. The third experiment is similar to the second one, except that the simulation is carried out *without* knowledge of the special party's secrets, using a standard/enhanced ECDSA oracle.

We show that our scheme is unforgeable by showing that if an environment forges signatures in an execution of our protocol, then the environment also forges signatures in all three experiments above, and from the third experiment we conclude that the environment forges signatures for the plain (non-threshold) ECDSA signature scheme, in contradiction with its presumed security.

The first two experiments are stepping stones towards proving that the environment forges in the third experiment. In more detail, the real execution and the first experiment are statistically close as long as all the ZK-proofs are sound (and the simulator extracts the right values). The first and the second experiment are computationally close as long as the Paillier cryptosystem is semantically secure. Finally, the second and the third experiment are identical (in a perfect sense).

Dealing w/ Adaptive Party Corruptions. To show that our protocol achieves security against adaptive party corruption, it is enough to argue that experiments 2 & 3 terminate. Assuming CDR and strong-RSA, our analysis yields that both experiments terminate in time quasi-proportional to the number of parties, and the environment forges signatures in the third experiment, in contradiction with the presumed security of plain ECDSA. Consequently, under suitable cryptographic assumptions, unless the environment corrupts all parties simultaneously in-between key-refresh phases, our scheme is unforgeable.

Overall UC-Security of our Protocol. From the above, it follows that if the ECDSA signature scheme is existentially unforgeable, then the online variant of our protocol UC-realizes the ideal signature functionality. Similarly, if ECDSA is *enhanced* existentially unforgeable, then the offline variant of our protocol UC-realizes the ideal signature functionality.

We remind the reader that existential unforgeability is defined via a game where a prospective forger is given access to a signing oracle allowing the attacker to sign (arbitrary) messages of his own choosing. The attacker wins the game if he manages to generate a valid signature for a previously unsigned message. We define an enhanced variant of the unforgeability game where the data of the signature that is independent of the message (i.e. $g^{k^{-1}}$, henceforth referred to as the signature's *nonce*) can be queried by the attacker *before* producing a message to be signed; that way the attacker can potentially choose messages for the signing oracle that are correlated with the random nonce, which may be useful towards generating a forgery.

Evidence for Enhanced Unforgeability. To support our assumption that ECDSA is *enhanced* existentially unforgeable, we show that it holds in the following idealized model:

1. In the random oracle model, as long as not too-many nonces are queried in advance, and standard (non-enhanced) ECDSA is existentially unforgeable.

2. In the random oracle and generic group model, unconditionally.[7]

Both of the above are shown via reduction. For Item 1, the reduction simulates the random oracle and attempts to guess the messages the adversary is going to request signatures for; this is why not too-many nonces may be queried in advance, since the guessing probability decreases (super) exponentially. For Item 2, the reduction simulates the group *as if* it were a free-group generated by two base-points G and X (corresponding to the group-generator and ECDSA public key, respectively). Since the simulated (free) group is indistinguishable from a generic group, it follows that any forgery exploits a weakness in the hash-function, which we rule out by assumption. These are captured by the following theorems:

Theorem 1.2 (Informal)**.** *In the ROM, any efficient attacker running in time τ winning the enhanced game with probability α yields an attacker running in time $O(\tau \cdot T^Q \log(T))$ that wins the standard game with probability α, where T is the number of calls to the random oracle, and $Q \in O(1)$ denotes the number signatures the attacker is allowed to request.*

Theorem 1.3 (Informal)**.** *In the ROM, any* generic *attacker wins the enhanced game with probability at most T/q, where q is the size of the field, and T is the number of calls to the random oracle.*

1.2.6 Non-Interactive Zero-Knowledge

Our protocol makes extensive use of Non-Interactive Zero-Knowledge (NIZK) via the standard technique of compiling three-move zero-knowledge protocols (also known as Σ-protocols) with the Fiat-Shamir transform (FS), i.e. the Verifier's messages are computed by the Prover himself by invoking a predetermined hash function.

In the random-oracle model, the Fiat-Shamir transform gives rise to NIZK proof-systems. Furthermore, because we completely avoid the need for "online extraction" (c.f. Section 1.2.5), our use of the Fiat-Shamir transform *does not* interfere universal composability, and our protocol is UC as described.

Range Proofs. We conclude the overview of our techniques by presenting a vanilla version of the (interactive) zero-knowledge technique we employ. The technique is somewhat standard [3, 7, 8, 23, 34]; we spell it out here for convenience. However, the analysis is somewhat complicated, and it is not crucial for understanding our threshold signature protocol. Thus the present section may be skipped, if so desired.

We recall that Paillier ciphertexts have the form $C = (1+N)^x r^N \mod N^2$, where N denotes the public key, $x \in \mathbb{Z}_N$ the plaintext, and r is a random element of $\mathbb{Z}_N^*$. We further recall the strong-RSA assumption: for an RSA modulus N of unknown factorization, for uniformly random $y \in \mathbb{Z}_N$, it is infeasible to find (x, e) such that $e > 1$ and $x^e = y \mod N$. Finally, before we describe the zero-knowledge technique, we (informally) define ring-Pedersen commitments.[8]

[7] To be more precise, we show that any generic forger finds x, y such that $\mathcal{H}(x)/\mathcal{H}(y) = e$, for a random $e \leftarrow \mathbb{F}_q$, where $\mathcal{H}$ denotes the hash-function. We conjecture that the latter is hard also for the actual implementation of ECDSA involving SHA.

[8] We use the prefix "ring" to distinguish between "group" Pedersen commitments which reside in groups of known order.

Definition 1.4 (Ring-Pedersen – Informal)**.** Let N be an RSA modulus and let $s,\ t \in \mathbb{Z}_N^*$ be non-trivial quadratic residues. A ring-Pedersen commitment of $m \in \mathbb{Z}_N$ with public parameters (N, s, t) is computed as $C = s^m t^\rho \mod N$ where $\rho \leftarrow \mathbb{Z}_N$.

Vanilla ZK Range-Proof. Consider the following relation:

$$R = \{(C_0, N_0, C_1, N_1, s, t; \alpha, \beta, r) \mid C_0 = (1+N_0)^\alpha r^{N_0} \mod N_0^2 \ \wedge\ C_1 = s^\alpha t^\beta \mod N_1 \ \wedge\ \alpha \in \pm 2^\ell\}.$$

In words, the Prover must show that the Paillier plaintext of C_0 is equal to the hidden value in the ring-Pedersen commitment C_1, and that it lies in the range $\pm 2^\ell = [-2^\ell, +2^\ell]$ where $2^\ell \ll N_0, N_1$. It is assumed that the Paillier modulus N_0 was generated by the Prover and the ring-Pedersen parameters (N_1, s, t) were generated by the Verifier. We further assume that N_0 and N_1 were generated as products of suitable[9] primes and that s and t are non-trivial quadratic residues in $\mathbb{Z}_{N_1}^*$. This assumption does not incur loss of generality, since in the actual protocol we instruct the parties to prove in zero-knowledge that all the parameters were generated correctly.[10]

We now turn to the description of the ZK-proof for the relation R under its interactive variant (the actual proof is compiled to be non-interactive using the Fiat-Shamir transform). We perform a Schnorr-type proof as follows: the Prover encrypts a random value γ as $D_0 = (1+N_0)^\gamma \rho^{N_0} \mod N_0^2$ for suitable random ρ, computes a ring-Pedersen commitment $D_1 = s^\gamma t^\delta \mod N_1$ to γ for suitable random δ, and sends (D_0, D_1) to the Verifier. The Verifier then replies with a challenge $e \leftarrow \pm 2^\ell$ and the Prover solves the challenge by sending $z_1 = \gamma + e\alpha$. The Verifier accepts only if z_1 is in a suitable range and passes two equality checks (one for the encryption and one for the commitment). Intuitively, the Prover cannot fool the Verifier because "the only way" for the Prover to cheat is knowing the order of $\mathbb{Z}_{N_1}^*$, which was secretly generated by the Verifier and therefore would violate the strong-RSA assumption. In more detail:

1. The Prover computes $D_0 = (1+N_0)^\gamma \rho^{N_0} \mod N_0^2$ and $D_1 = s^\gamma t^\delta \mod N_1$, for random elements $\gamma \leftarrow \pm 2^{\ell+\varepsilon}$, $\delta \leftarrow \pm N_1 \cdot 2^\varepsilon$ and $\rho \leftarrow \mathbb{Z}_{N_0}^*$, and sends (D_0, D_1) to the Verifier.

2. The Verifier replies with $e \leftarrow \pm 2^\ell$.

3. The Prover computes
$$\begin{cases} z_1 &= \gamma + e\alpha \\ z_2 &= \delta + e\beta \\ w &= \rho \cdot r^e \mod N_0 \end{cases}$$
and sends (z_1, z_2) to the Verifier.

- **Verification**: Accept if the $z_1 \in \pm 2^{\ell+\varepsilon}$ and $(1+N_0)^{z_1} w^N = C_0^e \cdot D_0 \mod N_0^2$ and $s^{z_1} t^{z_2} = C_1^e \cdot D_1 \mod N_1$.

We remark that there is a discrepancy between the range-check of z_1 and the desired range by a (multiplicative factor) of 2^ε, referred to as the slackness-parameter; this is a feature of the proof since the range of α is only guaranteed within that slackness-parameter. We now turn to the analysis of the ZK-proof (completeness, honest verifier zero-knowledge & soundness).

It is straightforward to show that the above protocol satisfies completeness and (honest-verifier) zero-knowledge with some statistical error. The hard task is showing soundness [8, 23, 34]. Following the standard paradigm, we show special soundness by extracting the secrets from two accepting transcripts of the form $(D_0, D_1, e, z_1, z_2, w)$ and $(D_0, D_1, e', z_1', z_2', w')$ such that $e \neq e'$. Let Δ_e, Δ_{z_1}, Δ_{z_2} denote the relevant differences. We observe that if Δ_e divides Δ_{z_1} and Δ_{z_2} (in the integers), then all the values can be extracted without issue as follows: α and β are set to $\Delta_{z_1}/\Delta_e \in \pm 2^{\ell+\varepsilon}$ and Δ_{z_2}/Δ_e, respectively, and ρ can be extracted from the equality $(w \cdot w'^{-1})^N = (C_0(1+N_0)^{-\alpha})^{\Delta_e} \mod N_0^2$ (which allows to compute a Δ_e-th root of w/w' modulo N_1 c.f. Fact D.2). Thus, the soundness-proof boils down to showing that Δ_e divides both Δ_{z_1} and

[9] N_0 and N_1 should be bi-primes obtained as products of safe primes.

[10] In reality, for efficiency reasons, we prove much weaker statements that are sufficient for our purposes.

Δ_{z_2}, *unless the strong-RSA problem is tractable.* Namely, there exists an algorithm $\mathcal{S}$ with black-box access to the Prover that can solve the strong-RSA challenge t (the second ring-Pedersen parameter).[11]

To elaborate further, it is assumed that $\mathcal{S}$ knows λ such that $t^\lambda = s \mod N_1$ and that λ is sampled from $[N_1^2]$ (and not just $[N_1]$). Thus, without getting too deep into the details, if $\Delta_e \nmid \Delta_{\boldsymbol{z}} = \lambda\Delta_{z_1} + \Delta_{z_2}$, then $\mathcal{S}$ can solve the strong-RSA challenge by computing Euclid's extended algorithm on Δ_e and $\Delta_{\boldsymbol{z}}$. On the other hand, if $\Delta_e \nmid \Delta_{z_1}$ or Δ_{z_2}, we claim that $\Delta_e \mid \Delta_{\boldsymbol{z}}$ with probability at most 1/2. To see why, observe that there exists at least another $\lambda' \neq \lambda$ in $[N_1^2]$ such that $t^\lambda = t^{\lambda'} = s \mod N_1$, because t has order $\phi(N_1)/4 = O(N_1)$ and λ was sampled uniformly in $[N_1^2]$. Since the Prover cannot distinguish between the two λ's (in a perfect information-theoretic sense), if $\Delta_e \nmid \Delta_{z_1}$ or Δ_{z_2}, then the probability that Δ_e divides $\lambda\Delta_{z_1} + \Delta_{z_2}$ is at most 1/2 (i.e. the Prover guessed correctly which of the λ's the algorithm $\mathcal{S}$ knows).[12] In conclusion, the probability that extraction fails is at most twice the probability of breaking strong-RSA, which is assumed to be negligible.

Removing the Computational Assumption in the ZK-Proof. We point at that there is a somewhat standard way [3, 4, 7] to tweak the above ZK-proof to obtain an unconditional extractor (that does not rely on strong-RSA or any other hardness assumption), at the expense of higher communication costs.[13] Consider the relation

$$R = \{(C_0, N_0; \alpha, r) \mid C_0 = (1 + N_0)^\alpha r^{N_0} \mod N_0^2 \ \wedge \ \alpha \in \pm 2^\ell\}.$$

Notice that it's the same as the previous relation except that we got rid of the ring-Pedersen commitment. Then, by removing D_1 and z_2 from the protocol above, *and restricting* $e \leftarrow \{0, 1\}$ (instead of $\pm 2^\ell$), we obtain a zero-knowledge proof of knowledge with unconditional extraction and soundness error 1/2. Using the same notation as before, notice that the new protocol guarantees that Δ_e divides Δ_{z_1} since $\Delta_e \in \{-1, 1\}$, and thus divisibility is guaranteed without any hardness assumption. On the downside, a malicious Prover may always cheat with probability 1/2 and thus the protocol must be repeated to achieve satisfactory soundness. Since the protocol involves Paillier operations, this would incur a rather expensive (super-logarithmic) blowup factor of the proof size.

1.2.7 Extension to t-out-of-n Access Structure

In this work we mainly focus on n-out-of-n multi-party signing, and do not explicitly consider the more general t-out-of-n threshold signing for $t < n$. Such a protocol can be derived almost immediately from our protocol herein for the online variant using Shamir secret-sharing, with relevant changes to the protocol's components, similarly to Gennaro and Goldfeder [24].

The same technique can also be applied for the non-interactive variant, but special care must be taken regarding the preprocessed data that the parties store in memory. Specifically, each distinct set of "authorized" parties (of size at least t) should generate fresh independent preprocessed data. A party taking part in different authorized sets must *not* use the same preprocessed data between the sets. We stress that signing two distinct messages using dependant shared preprocessed data can enable an attack revealing the private key.

1.3 Additional Related Work

All recent protocols for threshold ECDSA follow (variants) of the blueprint described in Section 1.2.1 where the parties locally generate shares $k_1^* \dots k_n^*$ of k [33, 21] or k^{-1} [24, 16] and then jointly computing $r = g^{k^{-1}}|_{x\text{-axis}}$ and shares of $k(m + rx)$ via a pairwise-multiplication protocol in combination with the masking technique described at the beginning of the present section. Furthermore, all protocols take a somewhat optimistic approach, where the correctness of the computed values in the multiplication is verified only after the computation takes place, and not during; this is the main source of the round-complexity cost.

Security-wise, as mentioned previously, Gennaro and Goldfeder [24] show that their protocol satisfies a game-based definition of security (i.e. unforgeability of their protocol) under standard assumptions (DDH, CDR, strong-RSA, ECDSA). The protocol of Castagnos et al. [16] follows the same template, except that it replaces Paillier with an encryption scheme based on class groups [15]. Specifically, they show that their

[11] Parameter t is not completely random in $\mathbb{Z}_{N_1}$ since it's a quadratic residue, but this does not affect the analysis.
[12] The argument is more subtle because we need to show that Δ_e cannot divide both values simultaneously (see Section 4.1).
[13] A similar trick in a different context appears in Lindell [32], from Boudot [3] and Brickell et al. [4]

scheme is unforgeable assuming DDH and additional assumptions on class groups of imaginary quadratic fields, specifically hard subgroup membership, low-order assumption and strong root.[14]

Lindell et al. [33] and Doerner et al. [21] show secure-function evaluation of the ECDSA functionality and prove that their respective protocols UC-realize said functionality in a hybrid model with ideal commitment and zero-knowledge, assuming DDH . However, as pointed out by the authors themselves, the practical subroutines they recommend to replace the ideal calls do not preserve Universal Composability (even in the ROM). We stress that our protocol satisfies Universal Composability in the ROM "out of the box".

2 Preliminaries

Notation. Throughout the paper $\mathbb{G}$ denotes a group of prime order q, and $\mathbb{F}_q$ the finite field with q elements. We let $\mathbb{Z}, \mathbb{N}$ denote the set of integers and natural number, respectively. We use sans-serif letters ($\mathsf{enc}, \mathsf{dec}, \ldots$) or calligraphic ($\mathcal{S}, \mathcal{A}, \ldots$) to denote algorithms. Secret values are always denoted with lower case letters ($p, q, \ldots$) and public values are *usually* denoted with upper case letters ($A, B, N, \ldots$). Furthermore, for a tuple of both public and secret values, e.g. an RSA modulus and its factors (N, p, q), we use a semi-colon to differentiate public form secret values (so we write $(N; p, q)$ instead of (N, p, q)). For $t \in \mathbb{Z}_N$, we write $\langle t \rangle = \{t^k \mod N \text{ s.t. } k \in \mathbb{Z}\}$ for the multiplicative group generated by t. For $\ell \in \mathbb{Z}$, we let $\pm\ell$ denote the interval of integers $\{-|\ell|, \ldots, 0, \ldots, |\ell|\}$. We write $x \leftarrow X$ for sampling x uniformly from a set X (or according to the distribution X). Finally, let $\gcd : \mathbb{N}^2 \to \mathbb{N}$ and $\phi : \mathbb{N} \to \mathbb{N}$ denote the gcd operation and Euler's phi function, respectively.

2.1 Definitions

Definition 2.1. We say that $N \in \mathbb{N}$ is a Paillier-Blum integer iff $\gcd(N, \phi(N)) = 1$ and $N = pq$ where p, q are primes such that $p, q \equiv 3 \mod 4$.

Definition 2.2 (Paillier Encryption)**.** Define the Paillier cryptosystem as the three tuple $(\mathsf{gen}, \mathsf{enc}, \mathsf{dec})$ below.

1. Let $(N; p, q) \leftarrow \mathsf{gen}(1^\kappa)$ where p and q are $\kappa/2$-long primes and $N = pq$. Write $\mathsf{pk} = N$ and $\mathsf{sk} = (p, q)$.
2. For $m \in \mathbb{Z}_N$, let $\mathsf{enc}_{\mathsf{pk}}(m; \rho) = (1 + N)^m \cdot \rho^N \mod N^2$, where $\rho \leftarrow \mathbb{Z}_N^*$.
3. For $c \in \mathbb{Z}_{N^2}$, letting $\mu = \phi(N)^{-1} \mod N$,

$$\mathsf{dec}_{\mathsf{sk}}(c) = \left(\frac{[c^{\phi(N)} \mod N^2] - 1}{N} \right) \cdot \mu \mod N.$$

Definition 2.3 (ECDSA)**.** Let $(\mathbb{G}, g, q)$ denote the group-generator-order tuple associated with a given curve. We recall that elements in $\mathbb{G}$ are represented as pairs $a = (a_x, a_y)$, where the a_x and a_y are referred to as the projection of a on the x-axis and y-axis respectively, denoted $a_x = a|_{x\text{-axis}}$ and $a_y = a|_{y\text{-axis}}$, respectively. The security parameter below is implicitly set to $\kappa = \log(q)$.

Parameters: Group-generator-order tuple $(\mathbb{G}, q, g)$ and hash function $\mathcal{H} : \boldsymbol{M} \to \mathbb{F}_q$.

1. $(X; x) \leftarrow \mathsf{gen}(\mathbb{G}, q, g)$ such that $x \leftarrow \mathbb{F}_q$ and $X = g^x$.
2. For $\mathrm{msg} \in \boldsymbol{M}$, let $\mathsf{sign}_x(m; k) = (r, k(m + rx)) \in \mathbb{F}_q^2$, where $k \leftarrow \mathbb{F}_q$ and $m = \mathcal{H}(\mathrm{msg})$ and $r = g^{k^{-1}}|_{x\text{-axis}} \mod q$.
3. For $(r, \sigma) \in \mathbb{F}_q^2$, define $\mathsf{vrfy}_X(m, \sigma) = 1$ iff $r = (g^m \cdot X^r)^{\sigma^{-1}}|_{x\text{-axis}} \mod q$.

2.2 NP-relations

Schnorr. For parameters $(\mathbb{G}, g)$ consisting of element g in group $\mathbb{G}$, the following relation verifies that the prover knows the exponent of the group-element X. For PUB_0 of the form $(\mathbb{G}, g)$, define

$$R_{\mathsf{sch}} = \{(\mathsf{PUB}_0, X; x) \mid X = g^x\}.$$

[14] These assumptions may be viewed as analogues of CDR & strong-RSA for class groups.

Paillier Encryption in Range. For Paillier public key N_0, the following relation verifies that the plaintext value of Paillier ciphertext C is in a desired range $\mathcal{I}$. Define

$$R_{\mathsf{enc}} = \left\{(N_0, \mathcal{I}, C; x, \rho) \mid x \in \mathcal{I} \ \wedge \ C = (1+N_0)^x \rho^{N_0} \in \mathbb{Z}^*_{N_0^2}\right\}.$$

Group Element vs Paillier Encryption in Range. For parameters $(\mathbb{G}, N)$ consisting of group $\mathbb{G}$ and Paillier-Blum Modulus N, the following relation verifies that the discrete logarithm of X base g is equal to the plaintext value of C and is in range $\mathcal{I}$. For PUB_1 of the form $(\mathbb{G}, N)$, define

$$R_{\mathsf{log}} = \left\{(\mathsf{PUB}_1, \mathcal{I}, C, X, g; x, \rho) \mid x \in \mathcal{I} \ \wedge \ C = (1+N)^x \rho^N \in \mathbb{Z}^*_{N^2} \ \wedge \ X = g^x\right\}.$$

Paillier Affine Operation with Group Commitment in Range. For parameters $(\mathbb{G}, g, N_0, N_1)$ consisting of element g and in group $\mathbb{G}$ and Paillier public keys N_0, N_1, the following relation verifies that a Paillier ciphertext $C \in \mathbb{Z}^*_{N_0^2}$ was obtained as an affine-like transformation on C_0 such that the multiplicative coefficient (i.e. ε) is equal to the exponent of $X \in \mathbb{G}$ in the range $\mathcal{I}$, and the additive coefficient (i.e. δ) is equal to the plaintext-value of $Y \in \mathbb{Z}_{N_1}$ and resides in the the range $\mathcal{J}$. For PUB_2 of the form $(\mathbb{G}, g, N_0, N_1)$, define $R_{\mathsf{aff\text{-}g}}$ to be all tuples $(\mathsf{PUB}_2, \mathcal{I}, \mathcal{J}, C, C_0, Y, X; \varepsilon, \delta, r, \rho)$ such that

$$(\varepsilon, \delta) \in \mathcal{I} \times \mathcal{J} \ \wedge \ C = C_0^\varepsilon \cdot (1+N_0)^\delta r^{N_0} \in \mathbb{Z}^*_{N_0^2} \ \wedge \ Y = (1+N_1)^\delta \rho^{N_1} \in \mathbb{Z}^*_{N_1^2} \ \wedge \ X = g^\varepsilon \in \mathbb{G}$$

Paillier Affine Operation with Paillier Commitment in Range. This is a variant of the previous relation, the only difference is that now ε is equal to the plaintext-value of $X \in \mathbb{Z}^*_{N_1^2}$ (rather than the exponent of $X \in \mathbb{G}$, as before). For PUB_3 of the form (N_0, N_1), define $R_{\mathsf{aff\text{-}p}}$ to be all tuples $(\mathsf{PUB}_3, \mathcal{I}, \mathcal{J}, C, C_0, Y, X; \varepsilon, \delta, r, \rho, \mu)$ such that

$$(\varepsilon, \delta) \in \mathcal{I} \times \mathcal{J} \ \wedge \ C = C_0^\varepsilon \cdot (1+N_0)^\delta r^{N_0} \in \mathbb{Z}^*_{N_0^2} \ \wedge \ Y = (1+N_1)^\delta \rho^{N_1} \in \mathbb{Z}^*_{N_1^2} \ \wedge \ X = (1+N_1)^\varepsilon \mu^{N_1} \in \mathbb{Z}^*_{N_1^2}$$

2.2.1 Auxiliary Relations

Paillier-Blum Modulus. The following relation verifies that a modulus N is coprime with $\phi(N)$ and is the product of exactly two suitable odd primes, where $\phi(\cdot)$ is the Euler function.

$$R_{\mathsf{mod}} = \{(N; p, q) \mid \mathsf{PRIMES} \ni p, q \equiv 3 \mod 4 \ \wedge \ N = pq \ \wedge \ \gcd(N, \phi(N)) = 1\}.$$

Ring-Pedersen Parameters. The following relation verifies that an element $s \in \mathbb{Z}^*_N$ belongs to the (multiplicative) group generated by $t \in \mathbb{Z}_N$.

$$R_{\mathsf{prm}} = \left\{(N, s, t; \lambda) \mid s = t^\lambda \mod N\right\}.$$

Remark 2.4. In what follows, to alleviate notation when no confusion arises, we omit writing the public parameters described by PUB_*.

2.3 Sigma-Protocols

In this section we define zero-knowledge protocols with focus on interactive three-move protocols, known as Σ-protocols. In Section 2.3.1, we compile these protocols using the random oracle via the Fiat-Shamir heuristic to generate (non-interactive) proofs. We define two notions of Σ-protocols. The first one is "non-extractable" zero-knowledge with standard soundness, i.e. for relation R and x such that there *does not* exist w satisfying $(x, w) \in R$, the probability that a cheating Prover convinces the Verifier that x satisfies the relation is negligible. The second definition augments the soundness property to enable extraction from two suitable accepting transcripts; the latter property is known as special soundness.

Definition 2.5. A Σ-protocol Π for relation R is a tuple $(\mathsf{P}_1, \mathsf{P}_2, \mathsf{V}_1, \mathsf{V}_2)$ of PPT algorithms such that

- P_1 takes input $\kappa = |x|$ and random input τ and outputs A, and V_1 outputs its random input e.

- P_2 takes input (x, w, τ, e) and outputs z, and V_2 takes input (x, A, e, z) and (deterministically) outputs a bit b.

Security properties:

– *Completeness.* If $(x, w) \in R$ then with overwhelming probability over the choice of $e \leftarrow \mathsf{V}_1$ (as a function of $|x|$), for every $A \leftarrow \mathsf{P}_1(\tau)$ and $z \leftarrow \mathsf{P}_2(x, w, \tau, e)$, it holds that $V_2(x, A, e, z) = 1$.

– *Soundness.* If x is false with respect to R (i.e $(x, w) \notin R$ for all w), then for any PPT algorithm P^* and every A, the following holds with overwhelming probability over $e \leftarrow \mathsf{V}_1$ (as a function of κ): If $z \leftarrow \mathsf{P}^*(x, A, e)$ then $V_2(x, A, e, z) = 0$.

– *HVZK.* There exists a simulator $\mathcal{S}$ such that $(A, e, z) \leftarrow \mathcal{S}(x)$ it holds and $V_2(x, A, e, z) = 1$ for every x, with overwhelming probability over the random coins of $\mathcal{S}$. Furthermore, the following distributions are statistically indistinguishable. For $(x, w) \in R$:

  * (A, e, z) where $e \leftarrow \mathsf{V}_1$ and $A \leftarrow \mathsf{P}_1(x, w, \tau)$, and $z = \mathsf{P}_2(x, w, \tau, e)$.
  * (A, e, z) where $e \leftarrow \mathsf{V}_1$ and $(A, z) \leftarrow \mathcal{S}(x, e)$.

We use Σ-protocols to prove that the Paillier-Blum modulus is well-formed (R_{mod}) and that the ring-Pedersen Parameters are suitable (R_{prm}), we denote these Σ-protocols Π^{mod} and Π^{prm}, respectively (c.f. Sections 4.3 and 4.4). Note that for Π^{mod} the first message A is empty, so we can assume that A is some constant default string.

Definition 2.6. A Σ-protocol Π_σ with setup σ and special soundness for relation R is a tuple $(\mathsf{S}, \mathsf{P}_1, \mathsf{P}_2, \mathsf{V}_1, \mathsf{V}_2)$ of PPT algorithms satisfying the same functionalities and security properties of the Σ-protocol definition (w/o setup and special soundness), with the following changes:

1. Setup algorithm S initially generates σ which is a common input to all other algorithms.

2. Soundness property is replaced with:

– *Special Soundness.* There exists an efficient extractor $\mathcal{E}$ such that for any x and P^* the following holds with overwhelming probability (over the choice of $\sigma \leftarrow \mathsf{S}$): If (A, e, z), $(A, e', z') \leftarrow \mathsf{P}^*(x, w, \sigma)$ such that $V_2(x, A, e, z) = V_2(x, A, e', z') = 1$ and $e \neq e'$, then for $w' \leftarrow \mathcal{E}(x, A, e, e', z, z')$ it holds that $(x, w') \in R$.

We remark that the Schnorr proof of knowledge (c.f. Appendix B.1) is a Σ-protocol with special soundness that does not take any setup parameter, and we denote the protocol Π^{sch} (note σ is omitted). By contrast, our protocols for R_{enc}, R_{log}, $R_{\mathsf{aff\text{-}g}}$ and $R_{\mathsf{aff\text{-}p}}$ (i.e. the range proofs) require a setup parameter in the form of an RSA modulus N and ring-Pedersen parameters $s, t \in \mathbb{Z}_N^*$ (c.f. Sections 4.1 and 4.2 and appendices B.2 and B.3), and we denote the respective protocols as $\Pi_\sigma^{\mathsf{enc}}$, $\Pi_\sigma^{\mathsf{log}}$, $\Pi_\sigma^{\mathsf{aff\text{-}g}}$ and $\Pi_\sigma^{\mathsf{aff\text{-}p}}$, respectively . However, our threshold signature protocol does not assume any trusted setup, and in reality the setup parameter is generated by the parties themselves (a different one for each party). We expand on this point next.

Generating the Setup Parameter for the Range Proofs. Looking ahead to the security analysis of our threshold signature protocol, we stress that although the above definition prescribes a trusted setup for $\sigma = (N, s, t)$, in actuality the setup parameter is generated by the Verifier (the intended recipient of the proof) and is a accompanied by a ZK-proof that N is well formed (using Π^{mod} and the compiler below) and that s, $t \in \mathbb{Z}_N^*$ are suitable (using Π^{prm} and the compiler below). In particular, the Prover generates distinct proofs (one for each Verifier using its personal σ) to prove the same statement x to multiple verifying parties.

Notation 2.7. In the sequel, we incorporate the setup parameter σ in the protocol description, and we write Π_j^* for the corresponding protocol using $\mathcal{P}_j$'s setup parameter (acting as the Verifier), for $* \in \{\mathsf{enc}, \mathsf{log}, \mathsf{aff\text{-}g}, \mathsf{aff\text{-}p}\}$, and we omit mentioning the "trusted" algorithm S.

2.3.1 ZK-Module

Next, we present how to compile the protocols above using a random oracle via the Fiat-Shamir heuristic. Namely, to generate a proof, the Prover computes the challenge e by querying the oracle on a suitable input, which incorporates the theorem and the first message. Then, completing the transcript by computing the last message with respect to e, and sending the transcript as the proof. Later, the Verifier accepts the proof if it is a valid transcript of the underlying Σ-protocol and e was well-formed (verified by querying the oracle as the Prover did).

Formally, we define the compiler via the ZK-Module from Figure 2. Notice that on top of the standard prove/verify operations, the ZK-module contains a commit operation for generating the first message $A \leftarrow \mathsf{P}_1$ of the ZK-Proof. This will be useful for the signature protocol later, and specifically for the security analysis that requires extraction, because we force the adversary to commit to the first message of the (future) proof.

FIGURE 2 (ZK-Module $\mathcal{M}$ for Σ-protocols)

Parameter: Hash Function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^h$.

- On input $(\texttt{com}, \Pi, 1^\kappa)$, interpret $\Pi = (\mathsf{P}_1, \ldots)$:
  sample τ from the prescribed domain, compute $A = \mathsf{P}_1(\tau, 1^\kappa)$ and output $(A; \tau)$.
- On input $(\texttt{prove}, \Pi, \mathsf{aux}, x; w, \tau)$, interpret $\Pi = (\mathsf{P}_1, \mathsf{P}_2, \ldots)$:
  compute $A = \mathsf{P}_1(\tau)$ and $e = \mathcal{H}(\mathsf{aux}, x, A)$ and $z = \mathsf{P}_2(x, w, \tau, e)$ and output (A, e, z).
- On input $(\texttt{vrfy}, \Pi, \mathsf{aux}, x, \psi)$, interpret $\Pi = (\ldots, \mathsf{V}_2)$ and $\psi = (A, e', z)$:
  output 1 if $\mathsf{V}_2(x, A, e', z) = 1$ and $e' = \mathcal{H}(\mathsf{aux}, x, A)$, and 0 otherwise.

Figure 2: ZK-Module $\mathcal{M}$ for Σ-protocols

The properties of completeness, zero-knowledge, soundness and special soundness are analogously defined for the resulting proof system.

Notation 2.8. Sometimes we omit writing the randomness τ in the tuple $(\texttt{prove}, \Pi, \mathsf{aux}, x; w, \tau)$, indicating that fresh randomness is sampled.

3 Protocol

Our ECDSA protocol consists of four phases; one phase for generating the (shared) key which is run once (Figure 5), one phase to refresh the secret key-shares and to generate the auxiliary information required for signing (i.e. Paillier keys & ring-Pedersen parameters – Figure 6), one to preprocess signatures before the messages are known (Figure 7), and, finally, one for computing signature-shares once the messages are known (Figure 8).

We present two variants for our protocol; one for online signing (Figure 3) and one for non-interactive signing (Figure 4). The two protocols are different only in how the aforementioned components are combined. Namely, for the online variant, the parties are instructed to run (sequentially) the presigning and signing phases every time a new signature is requested for some message known to all parties. For the offline variant, the presigning phase is ran ahead of time, before the message is known. Finally, for both protocols, the key generation is executed upon activation, and the auxiliary info and key-refresh phase is executed according to the key-refresh schedule.

Remark 3.1. Our protocol is parametrized by a hash function $\mathcal{H}$, which is invoked to obtain a hash-values in domains of different length (e.g the finite field with q elements or an ℓ-size stream of bits). Formally, this is captured by introducing multiple hash functions of varying length. However, to alleviate notation, we simply write $\mathcal{H}$ for each (separate) hash function.

FIGURE 3 (Threshold ECDSA: Online Signing)

- **Key-Generation:** Upon activation on input ($\texttt{keygen}, ssid, i$) from $\mathcal{P}_i$ do:
  1. Run the key generation phase from Figure 5 and obtain $(srid, \boldsymbol{X}, x_i)$.
  2. Run the auxiliary info. phase from Figure 6 on input $(\texttt{aux-info}, ssid, srid, \boldsymbol{X}, i)$ and do:
     - When obtaining output $(\boldsymbol{X}, \boldsymbol{N}, \boldsymbol{s}, \boldsymbol{t})$ and (x_i, p_i, q_i), set $sid = (ssid, srid, \boldsymbol{X}, \boldsymbol{N}, \boldsymbol{s}, \boldsymbol{t})$ and standby.
- **Signing:** On input $(\texttt{sign}, sid, \ell, i, \text{msg})$ from $\mathcal{P}_i$, do:
  1. Run the pre-signing phase from Figure 7 on input $(\texttt{pre-sign}, sid, 0, i)$.
  2. Set $m = \mathcal{H}(\text{msg})$ and run the signing phase from Figure 8 on input $(\texttt{sign}, sid, 0, i, m)$.
     - When obtaining output standby.
- **Key-Refresh:** On input $(\texttt{key-refresh}, ssid, srid, \boldsymbol{X}, i)$ from $\mathcal{P}_i$,
  1. Run the auxiliary info. phase from Figure 6 on input $(\texttt{aux-info}, ssid, srid, \boldsymbol{X}, i)$.
  2. Upon obtaining output $(\boldsymbol{X}, \boldsymbol{N}, \boldsymbol{s}, \boldsymbol{t})$ and (x_i, p_i, q_i), do:
     - Erase all pre-signing and auxiliary info of the form $(ssid, \ldots)$.
     - Reset $sid = (ssid, srid, \boldsymbol{X}, \boldsymbol{N}, \boldsymbol{s}, \boldsymbol{t})$ and standby.

Figure 3: Threshold ECDSA: Online Signing

FIGURE 4 (Threshold ECDSA: Non-Interactive Signing)

- **Key-Generation:** Same as in Figure 3.
- **Pre-Signing:** On input $(\texttt{pre-sign}, sid, L, i)$ from $\mathcal{P}_i$, do:
  1. Erase all pre-signing data $(ssid, \ldots)$.
  2. Run the pre-signing phase from Figure 7 concurrently on inputs $(\texttt{pre-sign}, sid, 1, i), \ldots, (\texttt{pre-sign}, sid, L, i)$.
     - When obtaining output standby.
- **Signing:** On input $(\texttt{sign}, sid, \ell, i, \text{msg})$ from $\mathcal{P}_i$, do:

  Set $m = \mathcal{H}(\text{msg})$ and run the signing phase from Figure 8 on input $(\texttt{sign}, sid, \ell, i, m)$.
  - When obtaining output standby.
- **Key-Refresh:** Same as in Figure 3.

Figure 4: Threshold ECDSA: Non-Interactive Signing

3.1 Key Generation

Next, we describe the key-generation phase. At its core, the key-generation consists of each party $\mathcal{P}_i \in \boldsymbol{P}$ sampling $x_i \leftarrow \mathbb{F}_q$ and sending the public-key share $X_i = g^{x_i}$ to all other parties, together with a Schnorr proof of knowledge of the exponent. The public key is then set to $X = \prod_j X_j$. For malicious security, we instruct the parties to commit (using the oracle) to their public-key share X_i *as well as* the first message A_i of the Schnorr proof. Thus, the adversary cannot influence the distribution of the private-key by choosing an X as a function of the honest parties' public key shares, and the adversary is committed to the first message of the Schnorr proof (i.e. A_i), which will be used to extract the witness later in the reduction.

Upon obtaining all the relevant values, if no inconsistencies were detected, set $X = \prod_j X_j$ and store the secret key-share x_i as well as the public key-shares $\boldsymbol{X} = (X_1, \ldots, X_n)$. For full details see Figure 5.

Remark 3.2. We observe that the protocol instructs the parties to (verifiably) broadcast some of their messages (as opposed to messages which are "sent to all", where equality verification is not required). For non-unanimous halting [28], this can be achieved in a point-to-point network using echo-broadcasting with one extra round of communication.

FIGURE 5 (ECDSA Key-Generation)

Round 1.

Upon activation on input $(\texttt{keygen}, ssid, i)$ from $\mathcal{P}_i$, interpret $ssid = (\ldots, \mathbb{G}, q, g, \boldsymbol{P})$, and do:

- Sample $x_i \leftarrow \mathbb{F}_q$ and set $X_i = g^{x_i}$.
- Sample $srid_i \leftarrow \{0,1\}^\kappa$ and compute $(A_i, \tau) \leftarrow \mathcal{M}(\texttt{com}, \Pi^{\mathsf{sch}})$.
- Sample $u_i \leftarrow \{0,1\}^\kappa$ and set $V_i = \mathcal{H}(ssid, i, srid_i, X_i, A_i, u_i)$.

Broadcast $(ssid, i, V_i)$.

Round 2.

When obtaining $(ssid, j, V_j)$ from all $\mathcal{P}_j$, send $(ssid, i, srid_i, X_i, A_i, u_i)$ to all.

Round 3.

1. Upon receiving $(ssid, j, srid_j, X_j, A_j, u_j)$ from $\mathcal{P}_j$, do:
   - Verify $\mathcal{H}(ssid, j, srid_j, X_j, A_j, u_j) = V_j$.
2. When obtaining the above from all $\mathcal{P}_j$, do:
   - Set $srid = \oplus_j srid_j$.
   - Compute $\psi_i = \mathcal{M}(\texttt{prove}, \Pi^{\mathsf{sch}}, (ssid, i, srid), X_i; x_i, \tau)$.

   Send $(ssid, i, \psi_i)$ to all $\mathcal{P}_j$.

Output.

1. Upon receiving $(ssid, j, \psi_j)$ from $\mathcal{P}_j$, interpret $\psi_j = (\hat{A}_j, \ldots)$, and do:
   - Verify $\hat{A}_j = A_j$.
   - Verify $\mathcal{M}(\texttt{vrfy}, \Pi^{\mathsf{sch}}, (ssid, j, srid), X_j, \psi_j) = 1$.
2. When passing above verification from all $\mathcal{P}_j$, output $X = \prod_j X_j$.

Errors. When failing a verification step or receiving a complaint from any other $\mathcal{P}_j \in \boldsymbol{P}$, report a complaint and halt.

Stored State. Store the following: $srid$, $\boldsymbol{X} = (X_1, \ldots, X_n)$ and x_i.

Figure 5: ECDSA Key-Generation

3.2 Key-Refresh & Auxiliary Information

At a very high-level, the auxiliary info. and key-refresh phase proceeds as follows. Each party $\mathcal{P}_i$ samples a Paillier modulus N_i obtained as a product of safe-primes, as well as ring-Pedersen parameters (s_i, t_i). Then,

FIGURE 6 (Auxiliary Info. & Key Refresh)

Round 1.

On input ($\texttt{aux-info}, sid, i$) from $\mathcal{P}_i$, do:

- Sample two 4κ-bit long safe primes (p_i, q_i). Set $N_i = p_i q_i$.
- Sample $x_i^1, \ldots, x_i^n \leftarrow \mathbb{F}_q$ subject to $\sum_j x_i^j = 0$. Set $X_i^j = g^{x_i^j}$, $\boldsymbol{Y}_i = (X_i^j)_j$, $\boldsymbol{x}_i = (x_i^j)_j$.
- Sample $r \leftarrow \mathbb{Z}_{N_i}^*, \lambda \leftarrow \mathbb{Z}_{\phi(N_i)}$, set $t_i = r^2 \mod N_i$ and $s_i = t_i^\lambda \mod N_i$.
- Sample $u_i \leftarrow \{0,1\}^\kappa$ and compute $V_i = \mathcal{H}(sid, i, \boldsymbol{Y}_i, u_i)$.
- Compute $\psi_i = \mathcal{M}(\texttt{prove}, \Pi^{\mathsf{mod}}, (sid, i), N_i; (p_i, q_i))$
- Compute $\psi_i' = \mathcal{M}(\texttt{prove}, \Pi^{\mathsf{mod}}, (sid, i, \psi_i), N_i; (p_i, q_i))$.
- Compute $\psi_i'' = \mathcal{M}(\texttt{prove}, \Pi^{\mathsf{prm}}, (sid, i), (N_i, s_i, t_i); \lambda)$.

Broadcast $(sid, i, V_i, N_i, s_i, t_i, \psi_i, \psi_i', \psi_i'')$.

Round 2.

1. Upon receiving $(sid, j, V_j, N_j, s_j, t_j, \psi_j, \psi_j', \psi_j'')$ from $\mathcal{P}_j$, do:
   - Verify $N_j \geq 2^{8\kappa}$.
   - Verify $\mathcal{M}(\texttt{vrfy}, \Pi^{\mathsf{mod}}, (sid, j), N_j, \psi_j) = 1$.
   - Verify $\mathcal{M}(\texttt{vrfy}, \Pi^{\mathsf{mod}}, (sid, j, \psi_j), N_j, \psi_j') = 1$.
   - Verify $\mathcal{M}(\texttt{vrfy}, \Pi^{\mathsf{prm}}, (sid, j), (N_j, s_j, t_j), \psi_j'') = 1$.
2. When passing above verification for all $\mathcal{P}_j$, do for every $\mathcal{P}_k$:
   - Sample $\rho_k \leftarrow \mathbb{Z}_{N_k}^*$, and set $C_i^k = \mathsf{enc}_k(x_i^k; \rho_k)$.
   - Compute $\psi_{j,i,k} = \mathcal{M}(\texttt{prove}, \Pi_j^{\mathsf{log}}, (sid, i), (\mathcal{I}_\varepsilon, C_i^k, X_i^k, g); (x_i^k, \rho_k))$ for every $\mathcal{P}_j$.

   Send $\big(sid, i, \boldsymbol{Y}_i, u_i, \big(\psi_{j,i,k}, C_i^k\big)_k\big)$ to each $\mathcal{P}_j$.

Output.

1. Upon receiving $\big(sid, j, \boldsymbol{Y}_j, u_j, \big(\psi_{i,j,k}, C_j^k\big)_k\big)$ from $\mathcal{P}_j$, do:
   - Verify $\prod_k X_j^k = \mathsf{id}_{\mathbb{G}}$.
   - Verify $\mathcal{H}(sid, j, \boldsymbol{Y}_j, u_j) = V_j$.
   - Verify $\mathcal{M}(\texttt{vrfy}, \Pi_i^{\mathsf{log}}, (sid, j), (\mathcal{I}_\varepsilon, C_j^k, X_j^k, g), \psi_{i,j,k}) = 1$ for every k.
2. When passing above verification for all $\mathcal{P}_j$, do:
   - Set $x_i^* = x_i + \sum_j \mathsf{dec}_i(C_j^i) \mod q$.
   - Set $X_k^* = X_k \cdot \prod_j X_j^k$ for every k.

   Output $(sid, i, \boldsymbol{X}^* = (X_k^*)_k, \boldsymbol{N} = (N_j)_j, \boldsymbol{s} = (s_j)_j, \boldsymbol{t} = (t_j)_j)$.

Errors. When failing a verification step or receiving a complaint from any other $\mathcal{P}_j \in \boldsymbol{P}$, report a complaint and halt.

Stored State. Store x_i^*, p_i, q_i.

Figure 6: Auxiliary Info. & Key Refresh

$\mathcal{P}_i$ samples a secret sharing $(x_i^1, \ldots, x_i^n)$ of $0 \in \mathbb{F}_q$, computes $\boldsymbol{Y}_i = (X_i^1 = g^{x_i^1}, \ldots, X_i^n = g^{x_i^n})$, and broadcasts $\boldsymbol{Y}_i, N_i, s_i, t_i$ to all. After receiving all the relevant values, party $\mathcal{P}_i$ encrypts each x_i^k under $\mathcal{P}_k$'s Paillier public key N_k (*including* his own) and obtains ciphertexts C_i^k, for all k, which he sends to all parties (the reasoning is explained below). Then, each $\mathcal{P}_i$ refreshes to a new private key-shares $x_i^* = x_i + \sum_\ell x_\ell^i \mod q$, updates public key-shares of all parties $X_j^* = X_j \cdot \prod_\ell X_\ell^j$, and stores new $(N_1, s_1, t_1), \ldots, (N_n, s_n, t_n)$. For malicious security, the aforementioned process is augmented with the following ZKP's:

(a) N_i is a Paillier-Blum Modulus.

(b) ZK-Proof that s_i belongs to the multiplicative group generated by t_i in $\mathbb{Z}_{N_i}^*$.

(c) The plaintext value of C_i^k modulo q is equal to the discrete logarithm of X_i^k.

Looking ahead to the security analysis, we point that our simulator extracts the Paillier keys of the malicious parties in Item (a) and we can thus extract all the secret values from the ciphertexts $\{C_i^k\}_{i,k}$ without issue. The steps described above are interleaved to obtain the two-round protocol from Figure 6.

3.3 Pre-Signing

We give a high-level overview of the pre-signing phase (Figure 7). Recall that at the end of the aux-info. phase, each party $\mathcal{P}_i$ has a Paillier encryption scheme $(\mathsf{enc}_i, \mathsf{dec}_i)$ with public key N_i, as well as ring-Pedersen parameters $s_i, t_i \in \mathbb{Z}_{N_i}$. Further recall that a ECDSA signature has the form $(r = g^{k^{-1}}|_{x\text{-axis}}, \sigma = k(m + rx))$ where $\mathcal{P}_i$ has an additive share x_i of x.

For comparison, we also recall that the gist of the G&G protocol [24]. The parties (jointly) compute a random point $g^{k^{-1}}$ together with local additive shares k_i, χ_i of k and $k \cdot x$, respectively. Further recall that $g^{k^{-1}}$ is obtained from $(g^\gamma)^{\delta^{-1}}$, for some jointly computed random value $\delta = k\gamma$, where γ is a (hidden) jointly generated mask for k. In more detail, the protocol proceeds as follows:

1. Each party $\mathcal{P}_i$ generates local shares k_i and γ_i, computes Paillier encryptions $K_i = \mathsf{enc}_i(k_i)$ and $G_i = \mathsf{enc}_i(\gamma_i)$, under $\mathcal{P}_i$'s key, and broadcasts (K_i, G_i).

2. For each $j \neq i$, party $\mathcal{P}_i$ samples $\beta_{i,j}, \hat{\beta}_{i,j} \leftarrow \mathcal{J}_\varepsilon$ and computes $D_{j,i} = \mathsf{enc}_j(\gamma_i \cdot k_j - \beta_{i,j})$ and $\hat{D}_{j,i} = \mathsf{enc}_j(x_i \cdot k_j - \hat{\beta}_{i,j})$ using the homomorphic properties of Paillier. Furthermore, $\mathcal{P}_i$ encrypts $F_{j,i} = \mathsf{enc}_i(\beta_{i,j})$, $\hat{F}_{j,i} = \mathsf{enc}_i(\hat{\beta}_{i,j})$, sets $\Gamma_i = g^{\gamma_i}$, and sends $(D_{j,i}, \hat{D}_{j,i}, F_{j,i}, \hat{F}_{j,i})$ to all parties.

3. Each $\mathcal{P}_i$ decrypts (and reduces modulo q) $\alpha_{i,j} = \mathsf{dec}_i(D_{i,j})$ and $\hat{\alpha}_{i,j} = \mathsf{dec}_i(\hat{D}_{i,j})$. and computes $\delta_i = \gamma_i \cdot k_i + \sum_{j \neq i} \alpha_{i,j} + \beta_{i,j} \mod q$, $\chi_i = x_i \cdot k_i + \sum_{j \neq i} \hat{\alpha}_{i,j} + \hat{\beta}_{i,j} \mod q$. Finally, $\mathcal{P}_i$ sets $\Gamma = \prod_j \Gamma_j$, $\Delta_i = \Gamma^{k_i}$ and sends δ_i, Δ_i to all parties.

When obtaining all δ_j's, party $\mathcal{P}_i$ sets $\delta = \sum_j \delta_j \mod q$ and verifies that $g^\delta = \prod_j \Delta_j$. If no inconsistencies are detected, $\mathcal{P}_i$ sets $R = \Gamma^{\delta^{-1}}$ and stores (R, k_i, χ_i). For malicious security, the aforementioned process is augmented with the following ZK-proofs:

(a) The plaintext of K_i lies in range $\mathcal{I}_\varepsilon$.

(b) The ciphertext $D_{j,i}$ was obtained as an affine-like opperation on K_j where the multiplicative coefficient is equal to the hidden value of G_i, and it lies in range $\mathcal{I}_\varepsilon$, and the additive coefficient is equal to hidden value of $F_{j,i}$, and lies in range $\mathcal{J}_\varepsilon$.

(c) The ciphertext $\hat{D}_{j,i}$ was obtained as an affine operation on K_j where the multiplicative coefficient is equal to the exponent of X_i, and it lies in range $\mathcal{I}_\varepsilon$, and the additive coefficient is equal to hidden value of $\hat{F}_{j,i}$, and it lies in range $\mathcal{J}_\varepsilon$.

(d) The exponent of Γ_i is equal to the plaintext-value of G_i.

Looking ahead to the security analysis, in order to simulate the protocol, it is enough to extract the k's, γ's, and β's of the adversary. Since the aforementioned values are encrypted under the malicious parties' Paillier keys, and the Paillier keys were extracted in previous phase, we can extract the desired values without issue.

FIGURE 7 (ECDSA Pre-Signing)

Recall that P_i's secret state contains x_i, p_i, q_i such that $X_i = g^{x_i}$ and $N_i = p_i q_i$.

Round 1.

On input $(\texttt{pre-sign}, sid, \ell, i)$ from $\mathcal{P}_i$, interpret $sid = (\ldots, \mathbb{G}, q, g, \boldsymbol{P}, srid, \boldsymbol{X}, \boldsymbol{N}, \boldsymbol{s}, \boldsymbol{t})$, and do:

- Sample $k_i, \gamma_i \leftarrow \mathbb{F}_q$, $\rho_i, \nu_i \leftarrow \mathbb{Z}^*_{N_i}$ and set $G_i = \mathsf{enc}_i(\gamma_i; \nu_i)$, $K_i = \mathsf{enc}_i(k_i; \rho_i)$.
- Compute $\psi^0_{j,i} = \mathcal{M}(\texttt{prove}, \Pi^{\mathsf{enc}}_j, (sid, i), (\mathcal{I}_\varepsilon, K_i); (k_i, \rho_i))$ for every $j \neq i$.

Broadcast (sid, i, K_i, G_i) and send $(sid, i, \psi^0_{j,i})$ to each $\mathcal{P}_j$.

Round 2.

1. Upon receiving $(sid, j, K_j, G_j, \psi^0_{i,j})$ from $\mathcal{P}_j$, do:
   - Verify $\mathcal{M}(\texttt{vrfy}, \Pi^{\mathsf{enc}}_i, (sid, j), (\mathcal{I}_\varepsilon, K_j), \psi_{i,j}) = 1$.
2. When passing above verification for all $\mathcal{P}_j$, set $\Gamma_i = g^{\gamma_i}$ and do:

   For every $j \neq i$, sample $r_{i,j}, s_{i,j}, \hat{r}_{i,j}, \hat{s}_{i,j} \leftarrow \mathbb{Z}_{N_j}$, $\beta_{i,j}, \hat{\beta}_{i,j} \leftarrow \mathcal{J}$ and compute:
   - $D_{j,i} = (\gamma_i \odot K_j) \oplus \mathsf{enc}_j(-\beta_{i,j}\ ,\ s_{i,j})$ and $F_{j,i} = \mathsf{enc}_i(\beta_{i,j}\ ,\ r_{i,j})$.
   - $\hat{D}_{j,i} = (x_i \odot K_j) \oplus \mathsf{enc}_j(-\hat{\beta}_{i,j}\ ,\ \hat{s}_{i,j})$ and $\hat{F}_{j,i} = \mathsf{enc}_i(\hat{\beta}_{i,j}\ ,\ \hat{r}_{i,j})$.
   - $\psi_{j,i} = \mathcal{M}(\texttt{prove}, \Pi^{\mathsf{aff\text{-}p}}_j, (sid, i), (\mathcal{I}_\varepsilon, \mathcal{J}_\varepsilon, D_{j,i}, K_j, F_{j,i}, G_i); (\gamma_i, \beta_{i,j}, s_{i,j}, r_{i,j}, \nu_i))$.
   - $\hat{\psi}_{j,i} = \mathcal{M}(\texttt{prove}, \Pi^{\mathsf{aff\text{-}g}}_j, (sid, i), (\mathcal{I}_\varepsilon, \mathcal{J}_\varepsilon, \hat{D}_{j,i}, K_j, \hat{F}_{j,i}, X_i); (x_i, \hat{\beta}_{i,j}, \hat{s}_{i,j}, \hat{r}_{i,j}))$.
   - $\psi'_{j,i} = \mathcal{M}(\texttt{prove}, \Pi^{\mathsf{log}}_j, (sid, i), (\mathcal{I}_\varepsilon, G_i, \Gamma_i, g); (\gamma_i, \nu_i))$.

   Send $(sid, i, \Gamma_i, D_{j,i}, F_{j,i}, \hat{D}_{j,i}, \hat{F}_{j,i}, \psi_{j,i}, \hat{\psi}_{j,i}, \psi'_{j,i})$ to each $\mathcal{P}_j$.

Round 3.

1. Upon receiving $(sid, j, \Gamma_j, D_{i,j}, F_{i,j}, \hat{D}_{i,j}, \hat{F}_{i,j}, \psi_{i,j}, \hat{\psi}_{i,j}, \psi'_{i,j})$ from $\mathcal{P}_j$, do
   - Verify $\mathcal{M}(\texttt{vrfy}, \Pi^{\mathsf{aff\text{-}p}}_i, (sid, j), (\mathcal{I}_\varepsilon, \mathcal{J}_\varepsilon, D_{i,j}, K_i, F_{j,i}, G_j), \psi_{i,j}) = 1$.
   - Verify $\mathcal{M}(\texttt{vrfy}, \Pi^{\mathsf{aff\text{-}g}}_i, (sid, j), (\mathcal{I}_\varepsilon, \mathcal{J}_\varepsilon, \hat{D}_{k,j}, K_i, \hat{F}_{j,i}, X_j), \hat{\psi}_{i,j}) = 1$.
   - Verify $\mathcal{M}(\texttt{vrfy}, \Pi^{\mathsf{log}}_i, (sid, j), (\mathcal{I}_\varepsilon, G_j, \Gamma_j, g), \psi'_{i,j}) = 1$.
2. When passing above verification for all $\mathcal{P}_j$, set $\Gamma = \prod_j \Gamma_j$ and $\Delta_i = \Gamma^{k_i}$, and do:
   - For every $j \neq i$, set $\alpha_{i,j} = \mathsf{dec}_i(D_{i,j})$ and $\hat{\alpha}_{i,j} = \mathsf{dec}_i(\hat{D}_{i,j})$ and

$$\begin{cases} \delta_i = \gamma_i k_i + \sum_{j \neq i} (\alpha_{i,j} + \beta_{i,j}) \mod q \\ \chi_i = x_i k_i + \sum_{j \neq i} (\hat{\alpha}_{i,j} + \hat{\beta}_{i,j}) \mod q \end{cases}.$$

   - For every $j \neq i$, compute $\psi''_{j,i} = \mathcal{M}(\texttt{prove}, \Pi^{\mathsf{log}}_j, (sid, i), (\mathcal{I}_\varepsilon, K_i, \Delta_i, \Gamma); (k_i, \rho_i))$.

   Send $(sid, i, \delta_i, \Delta_i, \psi''_{j,i})$ to each $\mathcal{P}_j$.

   Erase all items in memory except for the stored state.

Output.

1. Upon receiving $(sid, j, \delta_j, \Delta_j, \psi''_{i,j})$ from $\mathcal{P}_j$, do:
   - Verify $\mathcal{M}(\texttt{vrfy}, \Pi^{\mathsf{log}}_i, (sid, j), (\mathcal{I}_\varepsilon, K_j, \Delta_j, \Gamma), \psi''_{i,j}) = 1$.
2. When passing above verification for all $\mathcal{P}_j$, set $\delta = \sum_j \delta_j$, and do:
   - Verify $g^\delta = \prod_j \Delta_j$.
   - Set $R = \Gamma^{\delta^{-1}}$ and output (sid, i, R, k_i, χ_i).

     Erase all items except the stored state.

Errors. When failing a verification step or receiving a complaint from any other $\mathcal{P}_j \in \boldsymbol{P}$, report a complaint and halt.

Stored State. Store $\boldsymbol{X}$, $\boldsymbol{N}$, $\boldsymbol{s}$, $\boldsymbol{t}$ and (x_i, p_i, q_i).

Figure 7: ECDSA Pre-Signing

Preparing Multiple Signatures. To prepare L signatures, the parties follow the steps above L times concurrently. At the end of the presigning phase, each $\mathcal{P}_i$ stores the tuples $\{(\ell, R_\ell, k_{i,\ell}, \chi_{i,\ell})\}_{\ell\in[L]}$, and goes on standby.

Nota Bene. Recall that the public-key shares $\{X_i = g^{x_i}\}_{i\in[n]}$ are known to all parties, and let $\mathcal{I} = \pm 2^\ell$, $\mathcal{J} = \pm 2^{\ell'}$, $\mathcal{I}_\varepsilon = \pm 2^{\ell+\varepsilon}$ and $\mathcal{J}_\varepsilon = \pm 2^{\ell'+\varepsilon}$ denote integer intervals where ℓ, ℓ' and ε are fixed parameters (to be determined by the analysis). We represent integers modulo N in the interval $\{-N/2, \ldots, N/2\}$ (rather than the canonical representation); this convention is crucial to the security analysis.

3.4 Signing

Once the (hash of the) message m is known, on input $(\texttt{sign}, \ell, i, m)$ for the ℓ-th revealed point on the curve, the signing boils down to retrieving the relevant data and computing the right signature share. Namely, retrieve (ℓ, R, k, χ) compute $r = R|_{x\text{-axis}}$ and send $\sigma_i = km + r\chi \mod q$ to all. Erase the tuple (ℓ, R, k, χ). See Figure 8 for full details.

FIGURE 8 (ECDSA Signing)

Round 1.

On input $(\texttt{sign}, sid, \ell, i, m)$, if there is record of (sid, ℓ, R, k, χ), do:

- Set $r = R|_{x\text{-axis}}$ and $\sigma_i = km + r\chi$.
- Send (sid, i, σ_i) to all $\mathcal{P}_j$.

Erase (sid, ℓ, R, k, χ) from memory.

Output.

Upon receiving (sid, j, σ_j) from all $\mathcal{P}_j$, do:

- Set $\sigma = \sum_j \sigma_j$.
- Verify (r, σ) is a valid signature.

Output $(\texttt{signature}, sid, m, r, \sigma)$.

Errors. When failing a verification step or receiving a complaint from any other $\mathcal{P}_j \in \boldsymbol{P}$, report a complaint and halt.

Figure 8: ECDSA Signing

4 Underlying Σ-Protocols

We present the Σ-protocols associated with the NP-relations of Section 2.2. The Schnorr ZK-PoK as well as two of the protocols that are very similar to the ones below are moved to Appendix B.

4.1 Paillier Encryption in Range ZK (Π^{enc})

In Figure 9 we give a Σ-protocol for tuples of the form $(\mathcal{I} = \pm 2^\ell, C; k, r_0)$ satisfying relation R_{enc}. Namely, the Prover claims that he knows $k \in \pm 2^\ell$ such that $C = (1+N_0)^k \cdot r_0^{N_0} \mod N_0^2$. Let $(\hat{N}, s, t)$ be an auxiliary set-up parameter for the proof, i.e $\hat{N}$ is a suitable (safe bi-prime) Blum modulus and s and t are random squares in $\mathbb{Z}^*_{\hat{N}}$ (which implies $s \in \langle t \rangle$ with overwhelming probability).

Completeness. The protocol may reject a valid statement only if $|\alpha| \geq 2^{\ell+\varepsilon} - q2^\ell$ which happens with probability at most $q/2^\varepsilon$. □

Honest Verifier Zero-Knowledge. The simulator samples $z_1 \leftarrow \pm 2^{\ell+\varepsilon}$, $z_2 \leftarrow \mathbb{Z}^*_{N_0}$, $z_3 \leftarrow \pm \hat{N} \cdot 2^{\ell+\varepsilon}$, and $S \leftarrow \langle t \rangle$ by $S = t^\lambda \mod \hat{N}$ where $\lambda \leftarrow \pm 2^\ell \cdot \hat{N}$, and sets $A = (1+N_0)^{z_1} w^{N_0} \cdot K^{-e} \mod N_0^2$ and $C = s^{z_1} t^{z_3} \cdot S^{-e} \mod \hat{N}$. We observe that the real and simulated distributions are $2 \cdot q 2^{-\varepsilon} + 2^{-\ell} \approx 3q2^{-\varepsilon}$ statistically close (by

FIGURE 9 (Paillier Encryption in Range ZK – Π^{enc})

- **Setup:** Auxiliary RSA modulus $\hat{N}$ and Ring-Pedersen parameters $s, t \in \mathbb{Z}^*_{\hat{N}}$.
- **Inputs:** Common input is (N_0, K).
  The Prover has secret input (k, ρ) such that $k \in \pm 2^{\ell}$, and $K = (1+N_0)^k \cdot \rho^{N_0} \mod N_0^2$.

1. Prover samples

$$\alpha \leftarrow \pm 2^{\ell+\varepsilon} \text{ and } \begin{cases} \mu \leftarrow \pm 2^{\ell} \cdot \hat{N} \\ r \leftarrow \mathbb{Z}^*_{N_0} \\ \gamma \leftarrow \pm 2^{\ell+\varepsilon} \cdot \hat{N} \end{cases}, \text{ and computes } \begin{cases} S = s^k t^{\mu} \mod \hat{N} \\ A = (1+N_0)^{\alpha} \cdot r^{N_0} \mod N_0^2 \\ C = s^{\alpha} t^{\gamma} \mod \hat{N} \end{cases},$$

   and sends (S, A, C) to the Verifier.
2. Verifier replies with $e \leftarrow \pm q$
3. Prover sends (z_1, z_2, z_3) to the Verifier, where

$$\begin{cases} z_1 &= \alpha + ek \\ z_2 &= r \cdot \rho^e \mod N_0 \\ z_3 &= \gamma + e\mu \end{cases}.$$

- **Equality Checks**:

$$\begin{cases} (1+N_0)^{z_1} \cdot z_2^{N_0} = A \cdot K^e \mod N_0^2 \\ s^{z_1} t^{z_3} = C \cdot S^e \mod \hat{N} \end{cases}$$

- **Range Check**:

$$z_1 \in \pm 2^{\ell+\varepsilon}$$

  The proof guarantees that $k \in \pm 2^{\ell+\varepsilon}$.

Figure 9: Paillier Encryption in Range ZK – Π^{enc}

choosing $\ell = \varepsilon$ as we do in the analysis). This follows from Facts D.6, D.7, which imply z_1, z_3 are (each) $q2^{-\varepsilon}$ close to the real distribution, and S is $2^{-\ell}$ close to the real distribution. □

Special Soundness. Let $(S, A, C, e, z_1, z_2, z_3)$ and $(S, A, C, e', z_1', z_2', z_3')$ denote two accepting transcripts and let $(\Delta_e, \Delta_{z_1}, \Delta_{z_2}, \Delta_{z_3})$ denote the relevant differences. Notice that if Δ_e divides Δ_{z_1} and Δ_{z_3}(in the integers), then all the values can be extracted without issue as follows: k and μ are set to Δ_{z_1}/Δ_e and Δ_{z_3}/Δ_e. Finally, ρ can be extracted from the equality $(z_2/z_2')^{N_0} = ((1+N_0)^{-k} \cdot K)^{\Delta_e} \mod N_0^2$ and Fact D.2, *or*, using the factorization of N_0 in the case that $\Delta_e \mid N_0$, since N_0 is the product of exactly two primes. Therefore, it suffices to prove the claim below.

Claim 4.1 (Fujisaki and Okamoto [23], MacKenzie and Reiter [34])**.** *Assuming* sRSA*, it holds that* $\Delta_e \mid \Delta_{z_1}$ *and* $\Delta_e \mid \Delta_{z_3}$ *with probability at least* $1 - \mathsf{negl}(\kappa)$.

Define the predicate $\neg\mathsf{extract} \equiv (\Delta_e \not| \Delta_{z_1}) \vee (\Delta_e \not| \Delta_{z_3})$. We show that if $\neg\mathsf{extract}$ occurs with noticeable probability, then there is an algorithm $\mathcal{S}$ with black-box access to the Prover that can break sRSA with noticeable probability. More precisely, we show how to break sRSA as follows. The strong-RSA challenge is the second ring-Pedersen parameter t.[15] We assume that $\mathcal{S}$ knows $\lambda \in [\hat{N}^2]$ such that $s = t^{\lambda} \mod \hat{N}$, and λ is uniform in $[\hat{N}^2]$. We emphasize that the choice of $\hat{N}^2$ rather than $\hat{N}$ is crucial to the reduction.

Claim 4.2. *If* $\Delta_e \not| (\lambda\Delta_{z_1} + \Delta_{z_3})$*, then* sRSA *breaks.*

Proof of Claim 4.2. Define $\delta = \langle \lambda\Delta_{z_1} + \Delta_{z_3}, \Delta_e \rangle$ and let $\delta_e = \Delta_e/\delta$ and $\delta_z = (\lambda\Delta_{z_1} + \Delta_{z_3})/\delta$. Notice that $(S^{\nu_z} t^{\nu_e})^{\delta_e} = t \mod \hat{N}$, where (ν_e, ν_z) are the Bézout coefficients of δ_z and δ_e (i.e. $\nu_e\delta_e + \nu_z\delta_z = 1$),

[15] With probability 1/4, a uniform element in $\mathbb{Z}_{\hat{N}}$ is a random quadratic residue, and therefore computing non-trivial roots of t breaks sRSA, since t is a random quadratic residue.

since $S^{\delta_e} = t^{\delta_z}$. Deduce that the pair $(S^{\nu_z}t^{\nu_e}, \delta_e)$ is a successful response to the strong-RSA challenge, if $\Delta_e \nmid (\lambda\Delta_{z_1} + \Delta_{z_3})$. □

To conclude, we rule out (bound the probability) that $\Delta_e \mid \lambda\Delta_{z_1} + \Delta_{z_3}$ and $\neg\mathsf{extract}$; it suffices to bound the probability that $(\Delta_e \mid \lambda\Delta_{z_1} + \Delta_{z_3}) \wedge (\Delta_e \nmid \Delta_{z_1})$.[16] Write $\lambda = \lambda_0 + p_0q_0\lambda_1$, where $(p_0, q_0) = ((p-1)/2, (q-1)/2)$. Since Δ_z does not divide $p_0q_0\Delta_{z_1}$ (because $\langle \Delta_e, p_0q_0 \rangle = 1$ with overwhelming probability) we remark that, by Fact D.4, there exists a prime power a^b such that $a^b \mid p_0q_0\Delta_{z_1}$, $a^{b+1} \nmid \Delta_{z_1}$, and $\Delta_z = (\lambda_0\Delta_{z_1} + \Delta_{z_3}) + \lambda_1p_0q_0\Delta_{z_1} = 0 \mod a^{b+1}$ and thus λ_1 is uniquely determined modulo a. On the other hand, conditioned on the Prover's view, λ_1 has full entropy since $t^\lambda = t^{\lambda_0} \mod \hat{N}$, since t is a quadratic residue modulo $\hat{N}$, which means that, if $\Delta_e \nmid \Delta_{z_1}$, then the probability that $\Delta_e \mid \lambda\Delta_{z_1} + \Delta_{z_3}$ is at most $\frac{1}{a} + \mathsf{negl} \leq \frac{1}{2} + \mathsf{negl}$ over the Prover's coins, where the negligible term is of the form $(p+q) \cdot \mathsf{polylog}(\hat{N})/\hat{N}$. In conclusion, the probability that $\Delta_e \nmid \Delta_{z_1}$ or $\Delta_e \nmid \Delta_{z_3}$ is at most the probability of solving the RSA challenge divided by $(1/2 - \mathsf{negl})$, which is negligible overall. In more detail,

$$\begin{aligned}
\Pr[\neg\mathsf{extract}] &= \Pr[\Delta_e \mid (\lambda\Delta_{z_1} + \Delta_{z_3}) \wedge \neg\mathsf{extract}] + \Pr[\Delta_e \nmid (\lambda\Delta_{z_1} + \Delta_{z_3}) \wedge \neg\mathsf{extract}] \\
&= \Pr[\Delta_e \mid (\lambda\Delta_{z_1} + \Delta_{z_3}) \wedge \Delta_e \nmid \Delta_{z_1}] + \Pr[\mathsf{sRSA}] \\
&\leq (1/2 + \mathsf{negl}) \cdot \Pr[\Delta_e \nmid \Delta_{z_1}] + \Pr[\mathsf{sRSA}] \\
&\leq (1/2 + \mathsf{negl}) \cdot \Pr[\neg\mathsf{extract}] + \Pr[\mathsf{sRSA}]
\end{aligned}$$

□

4.2 Paillier Operation with Group Commitment in Range ZK ($\Pi^{\mathsf{aff\text{-}g}}$)

In Figure 10 we give a Σ-protocol for tuples of the form $(\mathcal{I} = \pm 2^\ell, \mathcal{J} = \pm 2^{\ell'}, C, Y, X; x, y, k, r_0)$ satisfying relation $R_{\mathsf{aff\text{-}g}}$. Namely, the Prover claims that he knows $x \in \pm 2^\ell$ and $y \in \pm 2^{\ell'}$ in range corresponding to group-element $X = g^x$ (on the curve) and Paillier ciphertext $Y = \mathsf{enc}_{N_1}(y) \in \mathbb{Z}^*_{N_1^2}$ and $C, D \in \mathbb{Z}^*_{N_0^2}$, such that $D = C^x(1+N_0)^y \cdot \rho^{N_0} \mod N_0^2$, for some $\rho \in \mathbb{Z}^*_{N_0}$. Let $(\hat{N}, s, t)$ be an auxiliary set-up parameter for the proof, i.e $\hat{N}$ is a suitable (safe bi-prime) Blum modulus and s and t are random squares in $\mathbb{Z}^*_{\hat{N}}$ (which implies $s \in \langle t \rangle$ with overwhelming probability).

Completeness. The protocol may reject a valid statement only if $|\alpha| \geq 2^{\ell+\varepsilon} - q2^\ell$ or $|\beta| \geq 2^{\ell'+\varepsilon} - q2^{\ell'}$ which happens with probability at most $q/2^{\varepsilon-1}$, by union bound. □

Honest Verifier Zero-Knowledge. The simulator samples $z_1 \leftarrow \pm 2^{\ell+\varepsilon}$, $z_2 \leftarrow \pm 2^{\ell'+\varepsilon}$, $z_3 \leftarrow \pm\hat{N} \cdot 2^{\ell+\varepsilon}$, $z_4 \leftarrow \pm\hat{N} \cdot 2^{\ell+\varepsilon}$, $w \leftarrow \mathbb{Z}^*_{N_0}$ and $S, T \leftarrow \langle t \rangle$ by $S = t^{\lambda_1} \mod \hat{N}, T = t^{\lambda_2} \mod \hat{N}$ where $\lambda_1, \lambda_2 \leftarrow \pm 2^\ell \cdot \hat{N}$, and sets $A = C^{z_1}(1+N_0)^{z_2}w^{N_0} \cdot D^{-e} \mod N_0^2$ and $B = g^{z_1}X^{-e} \in \mathbb{G}$ and $E = s^{z_1}t^{z_3} \cdot S^{-e} \mod \hat{N}$ and $F = s^{z_2}t^{z_4} \cdot T^{-e} \mod \hat{N}$. We observe that the real and simulated distributions are at most $4q \cdot 2^{-\varepsilon}$ far apart, by union bound and Facts D.6, D.7. □

Special Soundness. Let $(S, T, A, B, E, F, e, z_1, z_2, z_3, z_4, w, w_y)$ and $(S, T, A, B, E, F, e', z_1', z_2', z_3', z_4', w', w_y')$ denote two accepting transcripts such that $e \neq e'$ and let Δ_e, Δ_{z_1}, Δ_{z_2}, Δ_{z_3}, Δ_{z_4} denote the relevant differences. Similarly to the previous range proof, we show that Δ_e divides (over the integers $\mathbb{Z}$) each one of Δ_{z_1}, Δ_{z_2}, Δ_{z_3}, Δ_{z_4} and all the secrets can be extracted without issue. Using the same argument as in the previous proof, we observe that the probability that Δ_e does not divide Δ_{z_1} or Δ_{z_3} is at most $\Pr[\mathsf{sRSA}]/(\frac{1}{2} - \mathsf{negl}_1)$ and the probability that Δ_e does not divide Δ_{z_2} or Δ_{z_4} is at most $\Pr[\mathsf{sRSA}]/(\frac{1}{2} - \mathsf{negl}_2)$. Therefore, by union bound, we conclude that

$$\Pr[\neg\mathsf{extract}_1 \vee \neg\mathsf{extract}_2] \leq 2 \cdot \Pr[\mathsf{sRSA}] \cdot \left(\frac{1}{2} - \max(\mathsf{negl}_1, \mathsf{negl}_2)\right)^{-1}$$

where $\neg\mathsf{extract}_j$ denotes the event $(\Delta_e \nmid \Delta_{z_j} \vee \Delta_e \nmid \Delta_{z_{j+2}})$. □

[16] Since $\Delta_e \nmid \Delta_{z_3}$ and $\Delta_e \mid \lambda\Delta_{z_1} + \Delta_{z_3}$ implies $\Delta_e \nmid \Delta_{z_1}$.

FIGURE 10 (Paillier Affine Operation with Group Commitment in Range ZK – $\Pi^{\mathsf{aff\text{-}g}}$)

- **Setup:** Auxiliary Paillier Modulus $\hat{N}$ and Ring-Pedersen parameters $s, t \in \mathbb{Z}^*_{\hat{N}}$.
- **Inputs:** Common input is $(\mathbb{G}, g, N_0, N_1, C, D, Y, X)$ where $q = |\mathbb{G}|$ and g is a generator of $\mathbb{G}$.

  The Prover has secret input (x, y, ρ, ρ_y) such that $x \in \pm 2^{\ell}$, $y \in \pm 2^{\ell'}$, $g^x = X$, $(1+N_1)^y \rho_y^{N_1} = Y \mod N_1^2$, and $D = C^x(1+N_0)^y \cdot \rho^{N_0} \mod N_0^2$.

1. Prover samples $\alpha \leftarrow \pm 2^{\ell+\varepsilon}$ and $\beta \leftarrow \pm 2^{\ell'+\varepsilon}$ and

$$\begin{cases} r \leftarrow \mathbb{Z}^*_{N_0},\ r_y \leftarrow \mathbb{Z}^*_{N_1} \\ \gamma \leftarrow \pm 2^{\ell+\varepsilon} \cdot \hat{N},\ m \leftarrow \pm 2^{\ell} \cdot \hat{N} \\ \delta \leftarrow \pm 2^{\ell+\varepsilon} \cdot \hat{N},\ \mu \leftarrow \pm 2^{\ell} \cdot \hat{N} \end{cases} \quad \text{and computes} \quad \begin{cases} A = C^{\alpha} \cdot ((1+N_0)^{\beta} \cdot r^{N_0}) \mod N_0^2 \\ B_x = g^{\alpha} \in \mathbb{G} \\ B_y = (1+N_1)^{\beta} r_y^{N_1} \mod N_1^2 \\ E = s^{\alpha} t^{\gamma},\ S = s^x t^m \mod \hat{N} \\ F = s^{\beta} t^{\delta},\ T = s^y t^{\mu} \mod \hat{N} \end{cases}$$

   and sends (S, T, A, B, E, F) to the Verifier.
2. Verifier replies with $e \leftarrow \pm q$.
3. Prover Prover sends $(z_1, z_2, z_3, z_4, w, w_y)$ to the Verifier where

$$\begin{cases} z_1 = \alpha + ex \\ z_2 = \beta + ey \\ z_3 = \gamma + em \\ z_4 = \delta + e\mu \\ w = r \cdot \rho^e \mod N_0 \\ w_y = r_y \cdot \rho_y^e \mod N_1 \end{cases}$$

- **Equality Checks**:

$$\begin{cases} C^{z_1}(1+N_0)^{z_2} w^{N_0} = A \cdot D^e \mod N_0^2 \\ g^{z_1} = B_x \cdot X^e \in \mathbb{G} \\ (1+N_1)^{z_2} w_y^{N_1} = B_y \cdot Y^e \mod N_1^2 \\ s^{z_1} t^{z_3} = E \cdot S^e \mod \hat{N} \\ s^{z_2} t^{z_4} = F \cdot T^e \mod \hat{N} \end{cases}$$

- **Range Check**:

$$\begin{cases} z_1 \in \pm 2^{\ell+\varepsilon} \\ z_2 \in \pm 2^{\ell'+\varepsilon} \end{cases}$$

  The proof guarantees that $x \in \pm 2^{\ell+\varepsilon}$ and $y \in \pm 2^{\ell'+\varepsilon}$.

Figure 10: Paillier Affine Operation with Group Commitment in Range ZK – $\Pi^{\mathsf{aff\text{-}g}}$

4.3 Paillier-Blum Modulus ZK (Π^{mod})

In Figure 11 we give a Σ-protocol for tuples $(N; p, q)$ satisfying relation R_{mod}. The Prover claims that N is a Paillier-Blum modulus, i.e. $\gcd(N, \phi(N)) = 1$ and $N = pq$ where p, q are primes satisfying $p, q \equiv 3 \mod 4$. The following protocol is a combination (and simplification) of van de Graaf and Peralta [40] and Goldberg et al. [27].

FIGURE 11 (Paillier-Blum Modulus ZK – Π^{mod})

- **Inputs:** Common input is N. Prover has secret input (p, q) such that $N = pq$.

1. Prover samples a random $w \leftarrow \mathbb{Z}_N$ of Jacobi symbol -1 and sends it to the Verifier.
2. Verifier sends $\{y_i \leftarrow \mathbb{Z}_N\}_{i\in[m]}$
3. For every $i \in [m]$ set:
   - $x_i = \sqrt[4]{y_i'} \mod N$, where $y_i' = (-1)^{a_i} w^{b_i} y_i$ for unique $a_i, b_i \in \{0,1\}$ such that x_i is well defined.
   - $z_i = y_i^{N^{-1} \mod \phi(N)} \mod N$

   Send $\{(x_i, a_i, b_i), z_i\}_{i\in[m]}$ to the Verifier.

- **Verification:** Accept iff all of the following hold:
  - N is an odd composite number.
  - $z_i^N = y_i \mod N$ for every $i \in [m]$.
  - $x_i^4 = (-1)^{a_i} w^{b_i} y_i \mod N$ and $a_i, b_i \in \{0,1\}$ for every $i \in [m]$.

Figure 11: Paillier-Blum Modulus ZK – Π^{mod}

Completeness. Probability 1 by construction. □

Soundness. We first observe that the probability that y_i admits an N-th root if $\langle N, \phi(N)\rangle \neq 1$ is at most $1/\langle N, \phi(N)\rangle \leq 1/2$. Therefore, with probability 2^{-m}, it holds that $\langle N, \phi(N)\rangle = 1$, and, in particular, N is square-free. Next, if N is the product of more than 3 primes, the probability that $\{y_i, -y_i, wy_i, -wy_i\}$ contains a quadratic residue (which is necessary for being a quartic), for every i, is at most $(1/2)^m$, for any w.

On the other hand, if $N = pq$ and either q or $p \equiv 1 \mod 4$, then the probability that $\{y_i, -y_i, wy_i, -wy_i\}$ contains a quartic for every i is at most $(1/2)^{-m}$ for the following reason. Write $\mathcal{L} : \mathbb{Z}_N^* \mapsto \{-1,1\}^2$ such that $\mathcal{L}(x) = (a, b)$ where a is the Legendre symbol of x with respect to p and b is the Legendre symbol of x with respect to q. For fixed w, the table below upper bounds the probability that $\{y_i, -y_i, wy_i, -wy_i\}$ contains a quartic depending on the vallue of $\mathcal{L}(-1)$ and $\mathcal{L}(w)$; in red is the probability that it contains a square, and in blue is the probability that a random square is also a quartic, since the set contains exactly one square in those cases.

$\mathcal{L}(w) \setminus \mathcal{L}(-1)$	$(1,1)$	$(-1,1)$	$(1,-1)$	$(-1,-1)$
$(1,1)$	$1/4$	$1/2$	$1/2$	$1/2$
$(-1,1)$	$1/2$	$1/2$	$1/2$	$1/2$
$(1,-1)$	$1/2$	$1/2$	$1/2$	$1/2$
$(-1,-1)$	$1/2$	$1/2$	$1/2$	$1/2$

It follows that the probability that a square-free non-Blum modulus passes the above test is 2^{-m}, at most. Overall, the probability of accepting a wrong statement is at most 2^{-m+1}. □

Honest Verifier Zero-Knowledge. Sample a random γ_i and set $z_i' = \gamma_i^4$, and $x_i = \gamma_i^N$ and $y_i' = z_i'^N = x_i^4 \mod N$. Sample a random u with Jacobi symbol -1 and set $w = u^N \mod N$. Finally sample iid random bits $(a_i, b_i)_{i=1\ldots m}$ and do:

- For each $i \in [m]$, set $y_i = (-1)^{a_i} w^{-b_i} y_i'$ and $z_i = (-1)^{a_i} u^{-b_i} z_i'$
- Output $[w, \{y_i\}_i, \{(x_i, a_i, b_i), z_i\}_i]$.

Knowing that -1 is not a square modulo N with Jacobi symbol 1, the real and simulated distributions are identical. □

4.3.1 Extraction of Paillier-Blum Modulus Factorization

We stress that the above protocol is zero-knowledge only for *honest* Verifiers, which we strongly exploit in the security analysis of our threshold signature protocol. Specifically, assuming the Prover solves all challenges successfully, if the Verifier sends y_i's for which he secretly knows v_i such that $v_i^2 = (-1)^{a_i} w^{b_i} y_i \mod N$, then, for some i, the Verifier can deduce v_i' such that $v_i' \neq v_i, -v_i \mod N$ and $v_i'^2 = y_i \mod N$ with overwhelming probability. Thus, a malicious Verifier may efficiently deduce the factorization of N using the pair (v_i, v_i') (c.f. Fact D.5).

We strongly exploit the above in the security analysis our protocol. Specifically, when the adversary queries the random oracle to obtain a challenge for the ZK-proof that his Paillier-Blum modulus is well formed, the simulator programs the oracle accordingly in order to extract the factorization of the modulus. Namely:

Extraction. Sample random $\{v_i \leftarrow \mathbb{Z}_N\}_{i\in[m]}$ and iid bits $\{(a_i, b_i)\}_{i\in[m]}$ and set $y_i = (-1)^{a_i} w^{-b_i} v_i^2 \mod N$. Send $\{y_i\}_i$ to the Prover. If N is a Paillier-Blum modulus, then -1 is not a square modulo N with Jacobi symbol 1, and thus the y_i's are truly random, as long as w has Jacobi symbol -1. □

Remark 4.3. We point out that the extraction technique will only work if N is a Paillier-Blum modulus. This is the main reason why in the auxiliary info phase, we instruct the parties to "prove it twice". That way, we make sure that the modulus is Paillier-Blum, and then the simulator may accurately program the oracle to extract.

4.4 Ring-Pedersen Parameters ZK (Π^{prm})

The Σ-protocol of Figure 12 for the relation R_{prm} is a ZK-protocol for proving that s belongs to the multiplicative group of generated by t modulo N.

FIGURE 12 (Ring-Pedersen Parameters ZK – Π^{prm})

- **Inputs:** Common input is (N, s, t). Prover has secret input λ such that $s = t^\lambda \mod N$.

1. Prover samples $\{a_i \leftarrow \mathbb{Z}_{\phi(N)}\}_{i\in[m]}$ and sends $A_i = t^{a_i} \mod N$ to the Verifier.
2. Verifier replies with $\{e_i \leftarrow \{0,1\}\}_{i\in[m]}$
3. Prover sends $\{z_i = a_i + e_i\lambda \mod \phi(N)\}_{i\in[m]}$ to the Verifier.

- **Verification:** Accept if $t^{z_i} = A_i \cdot s^{e_i} \mod N$, for every $i \in [m]$.

Figure 12: Ring-Pedersen Parameters ZK – Π^{prm}

Completeness. Probability 1, by construction. □

Soundness. Suppose that $s \notin \langle t \rangle$. First observe that for any $z \in \phi(N)$, it holds that $s^{-1} \cdot t^z \notin \langle t \rangle$. Next notice that if $A \notin \langle t \rangle$, then $t^z \neq A \mod N$, for every z. It follows that the adversary generates an accepting transcript if he can guess correctly all the challenges, which happens with probability 2^{-m}. □

Zero-Knowledge. Sample $\{z_i \leftarrow \pm N/2\}_{i\in[m]}$ and $\{e_i \leftarrow \{0,1\}\}_{i\in[m]}$ and set $A_i = s^{-e_i} \cdot t^{z_i}$. The real and simulated distributions are statistically $m \cdot (1 - \phi(N)/N)$-close. □

Finally the Pedersen parameters can be generated as follows; sample $\tau \leftarrow \mathbb{Z}_N^*$ and $\lambda \leftarrow \mathbb{Z}_{\phi(N)}$ and set $t = \tau^2 \mod N$ and $s = t^\lambda \mod N$.

4.4.1 On the Auxilliary RSA moduli and the ring-Pedersen Parameters

The auxilliary moduli always belong to the Verifier and must be sampled as safe bi-prime RSA moduli. Furthermore, the pair (s, t) should consist of non-trivial quadratic residues in $\mathbb{Z}_{\hat{N}}$. In the actual setup, we sample $\hat{N}$ as a Blum (safe-prime product) integer and $s = \tau^{2d} \mod \hat{N}$ and $t = \tau^2 \mod N$ for a uniform $\tau \leftarrow \mathbb{Z}_{\hat{N}}$. During the auxiliary info phase, the (future) Verifier proves to the Prover that $s \in \langle t \rangle$.

The second issue which was implicitly addressed in the proofs above is how to sample uniform elements in $\langle t \rangle$. The naive idea is to sample random elements in $\phi(\hat{N})$ by sampling elements in $\hat{N}$. However, if $\hat{N}$ has small factors,[17] then small values close to zero will have noticeably more weight than other values, modulo $\phi(\hat{N})$. To fix this issue, we instruct the Prover (and the simulator in the proof of zero-knowledge) to sample elements from $\pm 2^{\ell} \cdot N$. That way, modulo $\phi(\hat{N})$, the resulting distribution is $\frac{1}{2^{\ell}}$-far from the uniform distribution in $\phi(N)$, by Fact D.7.

Choice of Moduli. With respect to our ECDSA protocol, for the Π^{enc} protocol, N_0 is the Paillier modulus of the Prover and and $\hat{N}$ is the Paillier modulus of the Verifier. And for the $\Pi^{\mathsf{aff\text{-}g}}$ protocol, $N_0, \hat{N}$ are the Paillier modulus of the Verifier, which is "reciever" of the homomorphic evaluation, and N_1 is the modulus of the Prover, which is the homomorphic "evaluator". Consult the Pre-Signing protocol at Figure 7 for all details.

5 Security Analysis

In this section we show that our protocol UC-realizes a proactive ideal threshold signature functionality ($\mathcal{F}_{\mathsf{sig}}$ from Figure 14). The present section presumes familiarity with the UC framework (see Appendix A for a brief overview). We adopt the random oracle model for our security analysis and we assume that all hash values (e.g. for the Fiat-Shamir Heuristic) are obtained by querying the random oracle, defined next.

5.1 Global Random Oracle

We use the definition of Canetti et al. [14], Camenisch et al. [9] for the ROM, parametrized by some output length h. We define the random oracle functionality such that when queried on a message m, the functionality returns a uniform output from a specified domain, and the functionality returns the same output for all future queries on m.

Real vs Ideal. The random oracle is peculiar in the security modelling. Specifically, the random oracle resides in both real and ideal processes giving the same answers in both worlds, and the parties may query it as prescribed and/or desired, *including* the environment.

FIGURE 13 (The Strict Global Random Oracle Functionality $\mathcal{H}^{\mathsf{g}}$)

Parameter: Output length h.

- On input $(\texttt{query}, m)$ from machine $\mathcal{X}$, do:
  - If a tuple (m, a) is stored, then output $(\texttt{answer}, a)$ to $\mathcal{X}$.
  - Else sample $a \leftarrow \{0,1\}^h$ and store (m, a).
    Output $(\texttt{answer}, a)$ to $\mathcal{X}$.

Figure 13: The Strict Global Random Oracle Functionality $\mathcal{H}^{\mathsf{g}}$

[17] If N has very small factors it's not an issue. The more problematic range of parameters is (as a function of the security parameter κ) $\hat{N} = \hat{p}\hat{q}$ where $q \sim \mathsf{poly}(\kappa)$ and $p \sim 2^{\kappa}/\mathsf{poly}(\kappa)$

5.2 Ideal Threshold Signature Functionality

Next, we describe our ideal threshold signature functionality. The functionality is largely an adaptation of the (non-threshold) signature functionality from Canetti [11], with an important addition to account for proactive security. See Figure 14 for the formal description of the functionality.

High-Level Description. When activated by all parties, the functionality requests a public key X and a verification algorithm $\mathcal{V}$ from the ideal-world adversary $\mathcal{S}$. Then, when all parties invoke the functionality to obtain a signature for some message m, the functionality requests a "signature" σ from $\mathcal{S}$ and records that σ is a valid signature for m. Finally, when the functionality is asked to verify some signature σ for a message m, the functionality either returns true/false if the pair (m, σ) is recorded as valid/invalid, or it applies the verification algorithm $\mathcal{V}$ and returns its output.

Proactive Security. The functionality is augmented with a corrupt/decorrupt and key-refresh interface capturing proactive security as follows: (1) the adversary may register parties as corrupted throughout the (ideal) process, (2) the adversary may decide to decorrupt parties, and those parties are recorded as "quarantined", and (3) if the key-refresh interface is activated, then the functionality erases all records of quarantined players. At any point in time, if the functionality records that all parties are corrupted/quarantined simultaneously, then the functionality effectively cedes control of the verification process to the adversary.

5.3 Security Claims

We show that our protocol UC-realizes functionality ($\mathcal{F}_{\mathsf{sig}}$ from Figure 14. Our proof follows by contraposition; under suitable cryptographic assumptions, we show that if our protocol does not UC-realize functionality $\mathcal{F}_{\mathsf{sig}}$, then there exists a PPT algorithm that can distinguish Paillier ciphertexts *or* there exists a PPT existential forger for the standard/enhanced ECDSA algorithm, in contradiction with the presumed security of the Paillier cryptosystem and the ECDSA signature scheme, respectively.

Theorem 5.1. *Assuming semantic security of the Paillier cryptosystem, strong-RSA assumption, and existential unforgeability of ECDSA, it holds that the protocol from Figure 3 UC-realizes functionality $\mathcal{F}_{\mathsf{sig}}$ from Figure 14.*

Theorem 5.2. *Assuming semantic security of the Paillier cryptosystem, strong-RSA assumption, and enhanced existential unforgeability of ECDSA, it holds that the protocol from Figure 4 UC-realizes functionality $\mathcal{F}_{\mathsf{sig}}$ from Figure 14.*

The rest of this section is dedicated to the analysis (simulators & proof) of Theorem 5.2. The analysis for Theorem 5.1 is essentially identical.

5.3.1 Proof of Theorem 5.2

Theorem 5.2 is a corollary of the following two lemmas.

Lemma 5.3. *If the protocol from Figure 4 does not UC-realize functionality $\mathcal{F}_{\mathsf{sig}}$, then there exists an environment $\mathcal{Z}$ that can forge signatures for previously unsigned messages in an execution of the protocol from Figure 4.*

Proof. The claim is immediate, since the ideal-process simulation is perfect (c.f. Section 5.4). □

Lemma 5.4. *The following holds assuming strong-RSA. If there exists an environment $\mathcal{Z}$ that can forge signatures for previously unsigned messages in an execution of the protocol from Figure 4, then there exists algorithms $\mathcal{R}_1$ and $\mathcal{R}_2$ with blackbox access to $\mathcal{Z}$ such that at least one of the items below is true.*

1. *$\mathcal{R}_1$ wins the semantic security experiment for Paillier with probability noticeably greater than $1/2$.*
2. *$\mathcal{R}_2$ wins the enhanced existential unforgeability experiment for (non-threshold) ECDSA with noticeable probability.*

FIGURE 14 (Ideal Threshold Signature Functionality $\mathcal{F}_{\mathsf{sig}}$)

Key-generation:

1. Upon receiving ($\texttt{keygen}, ssid$) from some party $\mathcal{P}_i$, interpret $ssid = (\ldots, \boldsymbol{P})$, where $\boldsymbol{P} = (\mathcal{P}_1, \ldots, \mathcal{P}_n)$.
   - If $\mathcal{P}_i \in \boldsymbol{P}$, send to $\mathcal{S}$ and record ($\texttt{keygen}, ssid, \mathcal{P}_i$).
   - Otherwise ignore the message.
2. Once ($\texttt{keygen}, ssid, j$) is recorded for all $\mathcal{P}_j \in \boldsymbol{P}$, send ($\texttt{pubkey}, ssid$) to the adversary $\mathcal{S}$ and do:
   (a) Upon receiving ($\texttt{pubkey}, ssid, X, \mathcal{V}$) from $\mathcal{S}$, record $(ssid, X, \mathcal{V})$.
   (b) Upon receiving ($\texttt{pubkey}, ssid$) from $\mathcal{P}_i \in \boldsymbol{P}$, output ($\texttt{pubkey}, ssid, X$) if it is recorded. Else ignore the message.

Signing:

1. Upon receiving ($\texttt{sign}, sid = (ssid, \ldots), m$) from $\mathcal{P}_i$, send to $\mathcal{S}$ and record ($\texttt{sign}, sid, m, i$).
2. Upon receiving ($\texttt{sign}, sid = (ssid, \ldots), m, j$) from $\mathcal{S}$, record ($\texttt{sign}, sid, m, j$) if $\mathcal{P}_j$ is corrupted. Else ignore the message.
3. Once ($\texttt{sign}, sid, m, i$) is recorded for all $\mathcal{P}_i \in \boldsymbol{P}$, send ($\texttt{sign}, sid, m$) to the adversary $\mathcal{S}$ and do:
   (a) Upon receiving ($\texttt{signature}, sid, m, \sigma$) from $\mathcal{S}$,
      - If the tuple $(sid, m, \sigma, 0)$ is recorded, output an error.
      - Else, record $(sid, m, \sigma, 1)$.
   (b) Upon receiving ($\texttt{signature}, sid, m$) from $\mathcal{P}_i \in \boldsymbol{P}$:
      - If $(sid, m, \sigma, 1)$ is recorded, output ($\texttt{signature}, sid, m, \sigma$) to $\mathcal{P}_i$.
      - Else ignore the message.

Verification:

Upon receiving ($\texttt{sig-vrfy}, sid, m, \sigma, X$) from a party $\mathcal{Q}$, send the tuple ($\texttt{sig-vrfy}, sid, m, \sigma, X$) to $\mathcal{S}$ and do:

- If a tuple (m, σ, β') is recorded, then set $\beta = \beta'$.
- Else, if m was never signed and not all parties in $\boldsymbol{P}$ are corrupted/quarantined, set $\beta = 0$. "*Unforgeability*"
- Else, set $\beta = \mathcal{V}(m, \sigma, X)$.

Record (m, σ, β) and output ($\texttt{istrue}, sid, m, \sigma, \beta$) to $\mathcal{Q}$.

Key-Refresh:

Upon receiving $\texttt{key-refresh}$ from $\mathcal{P}_i \in \boldsymbol{P}$, send $\texttt{key-refresh}$ to $\mathcal{S}$, and do:

- If not all parties in $\boldsymbol{P}$ are corrupted/quarantined, erase all records of ($\texttt{quarantine}, \ldots$).

Corruption/Decorruption:

1. Upon receiving ($\texttt{corrupt}, \mathcal{P}_j$) from $\mathcal{S}$, record $\mathcal{P}_j$ is corrupted.
2. Upon receiving ($\texttt{decorrupt}, \mathcal{P}_j$) from $\mathcal{S}$:
   - If not all parties are corrupted/quarantined do:
     If there is record that $\mathcal{P}_j$ is corrupted, erase it and record ($\texttt{quarantine}, \mathcal{P}_j$).
   - Else do nothing.

Figure 14: Ideal Threshold Signature Functionality $\mathcal{F}_{\mathsf{sig}}$

Proof of Lemma 5.4. Let $\mathcal{Z}$ denote environment that can forge signatures for previously unsigned messages in an execution of the protocol from Figure 4, and let $T \in \mathsf{poly}$ denote an upper bound on the number of times the auxiliary-info phase is ran before the forgery takes place. Let $N^1, \ldots, N^T$ and (X, x) denote Pallier public keys and ECDSA key-pair respectively, sampled according to the specifications of the protocol, and let $\mathcal{R}_1$ and $\mathcal{R}_2$ denote the processes from Sections 5.4.1 and 5.4.2, respectively. Consider the following three experiments:

Experiment A. Run $\mathcal{Z}$ with $\mathcal{R}_1$ on parameters (X, x) and $(N^k, c^k)_{k=1,\ldots,T}$ where $c^k = \mathsf{enc}_{N^k}(1)$.

Experiment B. Run $\mathcal{Z}$ with $\mathcal{R}_1$ on parameters (X, x) and $(N^k, c^k)_{k=1,\ldots,T}$ where $c^k = \mathsf{enc}_{N^k}(0)$.

Experiment C. Run $\mathcal{Z}$ with $\mathcal{R}_2$ on parameter X.

In words, process $\mathcal{R}_1$, dubbed the *Paillier-distinguisher*, simulates an interaction of the honest parties with the environment as follows. In the key-generation phase, $\mathcal{R}_1$ chooses the master-secret key x, and chooses the honest parties secret keys such that the master public key is equal to $X = g^x$ (this step requires rewinding the environment). Next, at the beginning of each key-refresh phase, $\mathcal{R}_1$ chooses a random honest party $\mathcal{P}_b$ and proceeds as follows. For all honest parties except $\mathcal{P}_b$, the simulation simply follows the instructions of the protocol. For $\mathcal{P}_b$, the simulation chooses Paillier keys drawn from $N^1 \ldots, N^T$ (viewed as a stack) and its messages are computed by (1) extracting the environments' secrets and (2) using the homomorphic properties of the Paillier cryptosystem. To elaborate further, we highlight that $\mathcal{R}_1$ takes as input a sequence of ciphertexts $c^1, \ldots, c^T$, because Paillier ciphertext encrypted under the special party's $\mathcal{P}_b$ key, say N^t, are computed as transformations on c^t, rather than as fresh encryptions. Furthermore, all of $\mathcal{P}_b$'s proofs are simulated using the relevant simulator and programming the oracle accordingly. Pre-signing and signing are simulated in a similar fashion.

Depending on the underlying plaintext value of c^t (either zero or one), the transcript of the interaction of $\mathcal{R}_1$ with $\mathcal{Z}$ is either "true", i.e. statistically close to the actual transcript of the real interaction between honest parties and environment, or is "fake" because all of the special party's ciphertexts are encryptions of zero. Finally, we remark that the special party's identity is rerandomized with every refresh-phase and the experiment is reset (by rewinding) to the last refresh, whenever the environment requests to corrupt the special party.

Claim 5.5. *Assuming strong-RSA, if $\mathcal{Z}$ outputs a forgery in an execution of the protocol from Figure 4 in time τ with probability α, then $\mathcal{Z}$ outputs a forgery in experiment A in time $\tau \cdot n \log(n)$ with probability at least $\alpha^2 - \mathsf{negl}(\kappa)$.*

Claim 5.6. *Assuming semantic security of the Paillier cryptosystem, if $\mathcal{Z}$ outputs a forgery in experiment A in time τ with probability α, then $\mathcal{Z}$ outputs a forgery in experiment B in time τ with probability at least $\alpha - \mathsf{negl}(\kappa)$.*

The second process $\mathcal{R}_2$, dubbed the *ECDSA-Forger*, simulates the interaction of the environment with the honest parties using only the public key and an enhanced signing oracle for plain (non-threshold) ECDSA, and it does not take any auxiliary input. The simulation proceeds as follows. In the key-generation phase, $\mathcal{R}_2$ chooses the honest parties' public keys such that the master public key is equal to X (this step requires rewinding the environment). To be more precise, the simulator chooses values as prescribed for all-but-one of the honest parties, and assigns public key share $X_b = X \cdot \prod_{j \neq qb} X_j$ for the randomly chosen special party. The remaining stages of the protocol are simulated in a similar fashion (by "compensating" for the unknown values using the special party) with the following important difference:

- The presigning simulation invokes the enhanced ECDSA oracle in order to obtain a point on the curve for future signing.
- The signing simulation requests signatures from the enhanced ECDSA oracle for points that were released earlier by the oracle.

Finally, similarly to the Paillier distinguisher, we remark that the special party's identity is rerandomized with every refresh-phase and the experiment is reset (by rewinding) to the last refresh, whenever the environment requests to corrupt the special party.

Claim 5.7. *If $\mathcal{Z}$ outputs a forgery in experiment B in time τ with probability α, then $\mathcal{Z}$ outputs a forgery in experiment C in time τ with probability α.*

□

5.4 Simulators

UC Simulator. As mentioned in the introduction, the description of the ideal-process adversary is essentially trivial. Namely, the simulator samples all values for the honest parties as prescribed, and follows the instructions of the protocol, for every phase. In broad strokes:

1. At the end of the key generation phase, the simulator sends the obtained public key X together with the ECDSA verification algorithm to the functionality.
2. At the end of each signing phase for some message msg, the simulator sends the computed signature (r, σ) to the functionality.
3. When the environment decides to corrupt/decorrupt a certain party, the simulator forwards the request to the functionality.

5.4.1 Paillier Distinguisher ($\mathcal{R}_1$)

The Paillier distinguisher $\mathcal{R}_1$ is parametrized by T and Paillier public keys and ciphertexts $N^1, \ldots, N^T$ and $C^1, \ldots, C^T$, and an ECDSA key-pair (X, x). Let ctr denote a counter variable initialized as $\mathsf{ctr} = 0$. Let $\boldsymbol{L}$ denote a list of query-answers that the simulator keeps in memory, initialized as an empty set. Algorithm $\mathcal{R}_1$ is defined by the following interaction with an environment $\mathcal{Z}$.

Oracle Calls.

Upon receiving $(\texttt{query}, m) = (\texttt{query}, ssid', srid', \ldots)$ from $\mathcal{Z}$, do:

1. If $(ssid', srid') \neq (ssid, srid)$ return $(\texttt{answer}, a = \mathcal{H}^{\mathsf{g}}(m))$.
2. Else if $m = ([sid, j, \psi], N)$ such that $\mathcal{M}(\texttt{vrfy}, \Pi^{\mathsf{mod}}, N, \psi) = 1$, then:
   - Program the oracle and extract p, q such that $N = pq$ (c.f. Section 4.3.1).
   - Add the relevant tuple to $\boldsymbol{L}$.
3. Else
   (a) If $(m, a) \in \boldsymbol{L}$, return $(\texttt{answer}, a)$.
   (b) Else sample a uniformly at random, return $(\texttt{answer}, a)$ and add (m, a) to $\boldsymbol{L}$.

Key-Generation.

The environment writes $(\texttt{keygen}, ssid = (\ldots, \boldsymbol{P}), i)$ on the input tape of $\mathcal{P}_i$, for each $\mathcal{P}_i$ and corrupts a strict subset of parties $\boldsymbol{C} \subsetneq \boldsymbol{P}$. Invoke $\mathcal{S}^1(ssid, \boldsymbol{C}, \boldsymbol{L}, X)$ and obtain output and obtain output b, $\boldsymbol{L}$, $srid$, $\{x_k\}_{k \neq b}$ and $\boldsymbol{X} = (X_1, \ldots)$. Set $x_b = x - \sum_{j \neq b} x_j \mod q$.

Aux-Info.

The environment writes $(\texttt{aux-info}, sid, \ell, i)$ of $\mathcal{P}_i$ and corrupts a strict subset of parties $\boldsymbol{C} \subsetneq \boldsymbol{P}$. Increment $\mathsf{ctr} = \mathsf{ctr} + 1$ and set $\mathsf{aux} = (\{x_i\}_{i \notin \boldsymbol{C}}, N^{\mathsf{ctr}}, C^{\mathsf{ctr}})$ Invoke $\mathcal{S}^2(sid, \boldsymbol{L}, \boldsymbol{C}, \mathsf{aux})$ and obtain output b and $\{N_j, s_j, t_j, (C_k^j)_k\}_{j \in \boldsymbol{P}}$ and $(p_i, q_i)_{i \in \boldsymbol{H}}$. Reassign $\{x_j = x_j + \sum_k \mathsf{dec}_j(C_k^j) \mod q\}_{k \neq b}$ and $x_b = x - \sum_{k \neq b} x_k \mod q$.

Presigning.

The environment writes $(\texttt{pre-sign}, sid, \ell, i)$ of $\mathcal{P}_i$ and corrupts a strict subset of parties $\boldsymbol{C} := \boldsymbol{C} \cup \boldsymbol{C}' \subsetneq \boldsymbol{P}$. Sample k_b and $\gamma_b \leftarrow \mathbb{F}_q$ and set $\boldsymbol{x}^{\setminus b} = (x_j)_{j \neq b}$ and $\mathsf{aux} = (c^{\mathsf{ctr}}, k_b, x_b, \gamma_b)$. Invoke $\mathcal{S}^3(sid, \boldsymbol{L}, \boldsymbol{C}, b, \boldsymbol{x}^{\setminus b}, \mathsf{aux})$ and obtain output $\{(sid, \ell, R, k_i, \chi_i)\}_{i \notin \boldsymbol{C}}$. .

Signing.

The environment writes $(\texttt{sign}, sid, \ell, m, i)$ of $\mathcal{P}_i$ and corrupts a strict subset of parties $\boldsymbol{C} := \boldsymbol{C} \cup \boldsymbol{C}' \subsetneq \boldsymbol{P}$.

1. Retrieve R and $\{(k_i, \chi_i)\}_{i \notin \boldsymbol{C}}$, set $r = R_\ell|_{x\text{-axis}}$.
2. Hand over $\{(sid, i, \sigma_i = k_i m + r\chi_i)\}_{i \notin \boldsymbol{C}}$.

Dynamic Corruptions.

- If $\mathcal{Z}$ corrupts $\mathcal{P}_i \in \boldsymbol{H}$, then reveal that party's (simulated) secret state.
- Else go back to $(\star)$ at the beginning of of the last invocation of simulator $\mathcal{S}^2$.

  Erase all items added to $\boldsymbol{L}$ since then.

5.4.2 ECDSA Forger ($\mathcal{R}_2$)

Our ECDSA forger $\mathcal{R}_2$ is parametrized by a public key X, and is defined by the following interaction with an environment $\mathcal{Z}$ and an enhanced ECDSA signing oracle for public key X. Let $\boldsymbol{L}$ denote a list of query-answers that the simulator keeps in memory, initialized as empty.

Oracle Calls.

Upon receiving $(\texttt{query}, m) = (\texttt{query}, ssid', srid', \ldots)$ from $\mathcal{Z}$, do:

1. If $(ssid', srid') \neq (ssid, srid)$ return $(\texttt{answer}, a = \mathcal{H}^{\mathrm{g}}(m))$.
2. Else if $m = ([sid, j, \psi], N)$ such that $\mathcal{M}(\texttt{vrfy}, \Pi^{\mathsf{mod}}, N, \psi) = 1$, then:
   - Program the oracle and extract p, q such that $N = pq$ (c.f. Section 4.3.1).
   - Add the relevant tuple to $\boldsymbol{L}$.
3. Else
   (a) If $(m, a) \in \boldsymbol{L}$, return $(\texttt{answer}, a)$.
   (b) Else sample a uniformly at random, return $(\texttt{answer}, a)$ and add (m, a) to $\boldsymbol{L}$.

Key-Generation.

The environment writes $(\texttt{keygen}, ssid = (\ldots, \boldsymbol{P}), i)$ on the input tape of $\mathcal{P}_i$, for each $\mathcal{P}_i$ and corrupts a strict subset of parties $\boldsymbol{C} \subsetneq \boldsymbol{P}$. Invoke $\mathcal{S}^1(ssid, \boldsymbol{C}, \boldsymbol{L}, X)$ and obtain output and obtain output b, $\boldsymbol{L}$, $srid$, $\{x_k\}_{k \neq b}$ and $\boldsymbol{X} = (X_1, \ldots)$.

Aux-Info.

The environment writes $(\texttt{aux-info}, sid = (ssid, srid, \ldots), i)$ of $\mathcal{P}_i$ and corrupts a strict subset of parties $\boldsymbol{C} \subsetneq \boldsymbol{P}$. Invoke $\mathcal{S}^2(sid, \boldsymbol{L}, \boldsymbol{C}, \mathsf{aux})$ and obtain output b and $\{N_j, s_j, t_j, (C_k^j)_k\}_{j \in \boldsymbol{P}}$ and $(p_i, q_i)_{i \notin \boldsymbol{C}}$. Reassign $\{x_j = \sum_k \mathsf{dec}_j(C_k^j)\}_{j \neq b}$ and $x_b = \perp$.

Presigning.

The environment writes $(\texttt{pre-sign}, sid, \ell, i)$ of $\mathcal{P}_i$ and corrupts a strict subset of parties $\boldsymbol{C} := \boldsymbol{C} \cup \boldsymbol{C}' \subsetneq \boldsymbol{P}$. Set $\boldsymbol{x}^{\setminus b} = (x_j)_{j \neq b}$, and do:

(a) Call the ECDSA oracle to obtain a point $R \in \mathbb{G}$. Sample $\delta \leftarrow \mathbb{F}_q$ and set $\mathsf{aux} = (R, \delta)$.
(b) Invoke $\mathcal{S}^3(sid, \boldsymbol{L}, \boldsymbol{C}, b, \boldsymbol{x}^{\setminus b}, \mathsf{aux})$ and obtain output $(sid, \ell, \eta^0, \eta^1)$ and $(sid, \ell, R, k_i, \chi_i)_{i \in \boldsymbol{H}}$.

Point-Reveal.

The environment writes $(\texttt{pt-rvl}, sid, \ell, i)$ on the input tape of $\mathcal{P}_i$ and corrupts $\boldsymbol{C} := \boldsymbol{C} \cup \boldsymbol{C}' \subsetneq \boldsymbol{P}$. The simulator proceeds as follows:

- Retrieve $(sid, \ell, \eta^0, \eta^1, \eta, \Gamma^*)$ and $(sid, \ell, \Gamma_i, \{G_j, D_j, \psi_{j,i} \ldots)_{i \in \boldsymbol{H}}$.
- Call the ECDSA oracle to obtain a point $R \in \mathbb{G}$.
- Choose $\delta \leftarrow \mathbb{F}_q$ and set $\Gamma_b = R^\delta \cdot (\Gamma^* \cdot \prod_{i \in \boldsymbol{H}} \Gamma_i)^{-1}$, and do:
  (a) For $\mathcal{P}_i \in \boldsymbol{H}$, hand over $(sid, i, \delta_i, \Gamma_i, \psi_{j,i}, \hat{\psi}_{j,i})$, for $j \neq i$.

(b) For $\mathcal{P}_b$, set $\delta_b = \delta - \eta$ and invoke ZK-simulator

$$\begin{cases} \psi_{j,b} \leftarrow \mathcal{S}^{\mathsf{dec}}_j(D_b, \ldots) \\ \hat{\psi}_{j,b} \leftarrow \mathcal{S}^{\mathsf{log}}(\Gamma_b, G_b, \ldots) \end{cases}$$

Add the relevant tuples to $\boldsymbol{L}$ and hand over $(sid, i, \delta_b, \Gamma_b, \psi_{j,b}, \hat{\psi}_{j,b})$ for $j \neq b$.

Record $(sid, \ell, R, \eta^0, \eta^1)$ and $(sid, \ell, R, k_j, \chi_j)_{i \in \boldsymbol{H}}$.

Signing.

The environment writes $(\mathtt{sign}, sid, \ell, m, i)$ on the input tape of $\mathcal{P}_i$ and corrupts $\boldsymbol{C} := \boldsymbol{C} \cup \boldsymbol{C}' \subsetneq \boldsymbol{P}$.

- Retrieve $(sid, \ell, R, \eta^0, \eta^1)$ and $(sid, \ell, R, k_i, \chi_i)_{i \in \boldsymbol{H}}$.
- Call the ECDSA oracle to sign m on point R to obtain signature (r, σ) and do:
  (a) For $\mathcal{P}_i \in \boldsymbol{H}$, compute σ_i as prescribed and hand over (sid, i, σ_i).
  (b) For $\mathcal{P}_b$, set $\sigma_b = \sigma - m\eta^0 - r\eta^1$ and and hand over (sid, b, σ_b).

Dynamic Corruptions.

- If $\mathcal{Z}$ corrupts $\mathcal{P}_i \in \boldsymbol{H}$, then reveal that party's (simulated) secret state.
- Else go back to $(\star)$ in round 2 of the auxiliary info simulator $\mathcal{S}^2$.

  Erase all items added to $\boldsymbol{L}$ since then.

5.5 Standalone Simulators

Notation 5.8. Write $\mathcal{S}^{\mathsf{sch}}, \mathcal{S}^{\mathsf{mul}}, \mathcal{S}^{\mathsf{dec}}_j, \mathcal{S}^{\mathsf{log}}_j, \mathcal{S}^{\mathsf{enc}}_j, \mathcal{S}^{\mathsf{aff}}_j$ for the ZK-simulators of $\Pi^{\mathsf{sch}}, \Pi^{\mathsf{mul}}, \Pi^{\mathsf{dec}}_j, \Pi^{\mathsf{log}}_j, \Pi^{\mathsf{enc}}_j, \Pi^{\mathsf{aff}}_j$.

5.5.1 Key-Generation Simulator ($\mathcal{S}^1$)

The simulator $\mathcal{S}^1(ssid, \boldsymbol{C}, \boldsymbol{L}, X)$ takes input the session identifier $ssid$, a list $\boldsymbol{L}$, a set of parties $\boldsymbol{C} \subsetneq \mathcal{P}$ and proceeds as follows.

Round 1.

- Initialize $\mathsf{ext} = 0$.
- Sample $\{V_i\}_{i \notin \boldsymbol{C}}$ in the prescribed domain and send $(ssid, i, V_i)$ to $\mathcal{Z}$, for each $\mathcal{P}_i \notin \boldsymbol{C}$.

Round 2.

(†) When obtaining $(ssid, j, V_j)$ for all $\mathcal{P}_j \in \boldsymbol{C}$,

1. If $\mathsf{ext} = 0$ compute all values as prescribed and hand over $\{(ssid, i, srid_i, X_i, A_i, u_i)\}_{i \notin \boldsymbol{C}}$ to $\mathcal{Z}$
2. Else choose $\mathcal{P}_b \leftarrow \boldsymbol{P} \setminus \boldsymbol{C}$ uniformly at random and let $\boldsymbol{H} = \boldsymbol{P} \setminus \boldsymbol{C} \cup \{\mathcal{P}_b\}$ and do:
   (a) For $\mathcal{P}_i \in \boldsymbol{H}$, sample all items as prescribed and hand over $(ssid, i, srid_i, X_i, A_i, u_i)$ to $\mathcal{Z}$.
   (b) For special party $\mathcal{P}_b$, set $X_b = X \cdot \prod_{j \neq b} X_j^{-1}$.

   Invoke ZK simulator $\psi_b = (A_b, \ldots) \leftarrow \mathcal{S}^{\mathsf{sch}}(X_b, \ldots)$.

   Hand over $(ssid, b, srid_b, X_b, A_b, u_b)$ to $\mathcal{Z}$, where $(srid_b, u_b)$ are sampled as prescribed.

   Add the relevant tuples to $\boldsymbol{L}$.

Round 3.

When obtaining all tuples $(ssid, j, srid_j, X_j, A_j, u_j)$,for every $\mathcal{P}_j \in \boldsymbol{C}$, add $\{\psi_j\}_{j \in \boldsymbol{C}}$ to $\boldsymbol{E}$ and do:

Set $srid = \oplus_j srid_j$ and hand over $\{(ssid, i, \psi_i)\}_{i \notin \boldsymbol{C}}$ to $\mathcal{Z}$. Add the relevant tuples to $\boldsymbol{L}$

Output.

1. If $\mathsf{ext} = 0$, set $\mathsf{ext} = 1$ and go back to (†) in round 2. Delete the pairs added to $\boldsymbol{L}$ since that point.
2. Else, extract $\{x_j\}_{j \notin C}$.

   Output b, $\boldsymbol{L}$, $srid$, $\{x_k\}_{k \neq b}$.

5.5.2 Auxiliary Info. & Key-Refresh Simulator ($\mathcal{S}^2$)

The auxiliary info. simulator $\mathcal{S}^2(sid, \boldsymbol{L}, \boldsymbol{C}, \mathsf{aux})$ takes input $sid = (ssid, srid, \ldots)$, a list $\boldsymbol{L}$, a set of parties $\boldsymbol{C} \subsetneq \boldsymbol{P}$, and auxiliary information $\mathsf{aux} = \perp$ or $\mathsf{aux} = (\{x_i\}_{i \notin \boldsymbol{C}}, N^*, C)$.

Round 1.

($\star$) Choose $\mathcal{P}_b \leftarrow \boldsymbol{P} \setminus \boldsymbol{C}$ uniformly at random and set $\boldsymbol{H} = \boldsymbol{P} \setminus \boldsymbol{C} \cup \{\mathcal{P}_b\}$.

1. For each $\mathcal{P}_i \in \boldsymbol{C}$, do:

   Sample all items as prescribed and hand over $(sid, i, N_i, s_i, t_i, V_i, \psi_i, \hat{\psi}_i, \psi'_i)$ to $\mathcal{Z}$.

2. For $\mathcal{P}_b$, do:
   (a) If $\mathsf{aux} = \perp$, sample $(N_b, p_b, q_b, s_b, t_b)$ as prescribed and V_b uniformly at random.
   (b) If $\mathsf{aux} \neq \perp$, set $N_b = N^*$ and sample (s_b, t_b) as prescribed and V_b uniformly at random. Invoke

$$\begin{cases} \psi_b \leftarrow \mathcal{S}^{\mathsf{mod}}(N_b, \ldots) \\ \psi'_b \leftarrow \mathcal{S}^{\mathsf{mod}}(N_b, \ldots) \\ \psi''_b \leftarrow \mathcal{S}^{\mathsf{prm}}(s_b, t_b, \ldots) \end{cases}$$

   Hand over $(sid, b, N_b, s_b, t_b, V_b, \psi_b, \psi'_b, \psi''_b)$ to $\mathcal{Z}$.

   Add the relevant pairs for the corresponding oracle calls to $\boldsymbol{L}$.

Round 2.

When obtaining all tuples $(sid, j, V_j, N_j, s_j, t_j, \psi_j, \psi'_j, \psi''_j)$, for every $\mathcal{P}_j \in \boldsymbol{C}$, do:

For each $\mathcal{P}_i \notin \boldsymbol{C}$, do:

(a) If $\mathsf{aux} \neq \perp$,
  - Sample $\{x_i^k\}_k$ uniformly subject to $0 = \sum_k x_i^k$ and set $\{X_i^k = g^{x_i^k}\}_k$.
  - Set $\{C_i^k = \mathsf{enc}_k(x_i^k)\}_{k \neq b}$ and $C_i^b = C^{x_i^b} \cdot \mathsf{enc}_b(0)$.
  - Invoke the ZK-simulator $\{\psi_{j,i,b} \leftarrow \mathcal{S}_j^{\mathsf{log}}(C_i^b, g, X_i^b \ldots)\}_{j \neq i}$, compute all other proofs as prescribed.

(b) If $\mathsf{aux} = \perp$,
  - Sample $\{x_i^k\}_{k \neq b}$ and set $\{X_i^k = g^{x_i^k}\}_{k \neq b}$ and $X_i^b = \mathsf{id}_{\mathbb{G}} \cdot g^{-\sum_{k \neq b} x_i^k}$.
  - Set $\{C_i^k = \mathsf{enc}_k(x_i^k)\}_{k \neq b}$ and $C_i^b = \mathsf{enc}_b(0)$.
  - Invoke the ZK-simulator $\{\psi_{j,i,b} \leftarrow \mathcal{S}_j^{\mathsf{log}}(C_i^b, X_i^b, \ldots)\}_{j \neq i}$, compute all other proofs as prescribed.

Sample u_i uniformly at random and hand over the tuple $(sid, i, \boldsymbol{Y}_i, u_i, \{\psi_{j,i,k}, C_i^k\}_k)$ for each $j \in \boldsymbol{P}$.

Add the revant tuples to $\boldsymbol{L}$.

Output.

Output $\{N_j, s_j, t_j, (C_k^j)_k\}_{j \in \boldsymbol{P}}$ and $(p_j, q_j)_{j \notin \boldsymbol{P}}$, where p_b, q_b are defined only if $\mathsf{aux} = \perp$.

5.5.3 Pre-Signing Simulator ($\mathcal{S}^3$)

The pre-signing simulator $\mathcal{S}^3(sid, \boldsymbol{L}, \boldsymbol{C}, b, \boldsymbol{x}^{\setminus b}, \mathsf{aux})$ takes inputs $sid = (\ldots, \boldsymbol{P}, \boldsymbol{X}, \boldsymbol{N}, \boldsymbol{s}, \boldsymbol{t})$, a list $\boldsymbol{L}$ and a set of parties $\boldsymbol{C} \subsetneq \boldsymbol{P}$, an index b and $\boldsymbol{x}^{\setminus b} = (x_j)_{j \neq b}$ such that $\mathcal{P}_b \notin \boldsymbol{C}$ and $g^{x_j} = X_j$ for $j \neq b$, and auxiliary information $\mathsf{aux} = (R, \delta)$ or $\mathsf{aux} = (c, x_b, k_b, \gamma_b)$.

Round 1.

1. For $\mathcal{P}_i \in \boldsymbol{H}$, compute all items as prescribed and hand over $(sid, i, K_i, G_i, \psi^0_{j,i})$ to $\mathcal{Z}$.

2. For $\mathcal{P}_b$, sample set

$$K_b = \begin{cases} c^{k_b} \cdot \mathsf{enc}_b(0) & \text{if } \mathsf{aux} \neq \perp \\ \mathsf{enc}_b(0) & \text{otherwise} \end{cases}$$

$$G_b = \begin{cases} c^{\gamma_b} \cdot \mathsf{enc}_b(0) & \text{if } \mathsf{aux} \neq \perp \\ \mathsf{enc}_b(0) & \text{otherwise} \end{cases}$$

Invoke the ZK-simulators $\psi^0_{j,b} \leftarrow \mathcal{S}^{\mathsf{enc}}_j(K_b, \ldots)$.

Hand over $(sid, b, K_b, G_b, \psi^0_{j,b})$ to $\mathcal{Z}$ and add the relevant tuples to $\boldsymbol{L}$.

Round 2.

- Upon receiving $(sid, j, K_j, G_j, \ldots)$ retrieve (k_j, γ_j)
- When obtaining all relevant tuples, do:
  1. For $\mathcal{P}_i \in \boldsymbol{H}$, send the tuple $(sid, i, \Gamma_i, D_{j,i}, F_{j,i}, \hat{D}_{j,i}, \hat{F}_{j,i}, \psi_{j,i}, \hat{\psi}_{j,i}, \psi'_{j,i})$ to $\mathcal{Z}$, for each $j \neq i$, where all values are computed as prescribed.
  2. For $\mathcal{P}_b$, sample $\{(\alpha_{\ell,b}, \hat{\alpha}_{\ell,b} \leftarrow \mathcal{J}^2\}_{\ell \neq b}$ and set $\hat{D}_{\ell,b} = \mathsf{enc}_\ell(\hat{\alpha}_{\ell,b})$ and $D_{\ell,b} = \mathsf{enc}_j(\alpha_{\ell,b})$, and

$$\hat{F}_{b,\ell} = \begin{cases} c^{k_\ell x_b - \hat{\alpha}_{\ell,b}} \cdot \mathsf{enc}_b(0) & \text{if } x_b \neq \perp \\ \mathsf{enc}_b(0) & \text{otherwise} \end{cases}$$

$$F_{b,\ell} = \begin{cases} c^{k_\ell \gamma_b - \alpha_{\ell,b}} \cdot \mathsf{enc}_b(0) & \text{if } \gamma_b \neq \perp \\ \mathsf{enc}_b(0) & \text{otherwise} \end{cases}$$

$$\Gamma_b = \begin{cases} g^{\gamma_b} & \text{if } \gamma_b \neq \perp \\ R^\delta \cdot g^{-\sum_{j \neq b} \gamma_j} & \text{otherwise} \end{cases}$$

Then, for each $j \neq b$, invoke the ZK-simulator

$$\begin{cases} \psi_{j,b} \leftarrow \mathcal{S}^{\mathsf{aff}}_j(D_{j,b}, K_j, \ldots), \\ \hat{\psi}_{j,b} \leftarrow \mathcal{S}^{\mathsf{aff}}_j(\hat{D}_{j,b}, K_j, \ldots) \\ \psi'_{j,b} \leftarrow \mathcal{S}^{\mathsf{log}}_j(\Gamma_b, g, G_b, \ldots) \end{cases}$$

Hand over the tuple $(sid, b, \Gamma_b, D_{j,b}, F_{j,b}, \hat{D}_{j,b}, \hat{F}_{j,b}, \psi_{j,b}, \hat{\psi}_{j,b}, \psi'_{j,b})$, for each $j \neq b$.

Add the relevant pairs for the corresponding oracle calls to $\boldsymbol{L}$.

Round 3. Upon receiving all $(sid, j, \Gamma_j, D_{i,j}, F_{i,j}, \hat{D}_{i,j}, \hat{F}_{i,j}, \psi_{i,j}, \hat{\psi}_{i,j}, \psi'_{i,j})$ for $j \neq i$, do:

1. If $\mathsf{aux} = (R, \delta)$, Set $\Delta_b = g^\delta \cdot \prod_{j \neq b} \Gamma^{k_j}$ and set

$$\begin{cases} \eta^0 = \sum_{j \neq b} k_j \\ \eta^1 = \sum_{j,i \neq b} k_i x_j + \sum_{j \neq b} \hat{\alpha}_{j,b} + \hat{\beta}_{j,b} \\ \delta_b = \delta - \sum_{j \neq b} \alpha_{j,b} + \beta_{j,b} + \sum_{i,j \neq b} k_i \gamma_j \end{cases}$$

Invoke ZK-simulator $\psi''_{j,b} \leftarrow \mathcal{S}^{\mathsf{log}}_j(\Delta_b, \Gamma, K_b, \ldots)$, for $j \neq b$.

Hand over $\{(sid, i, \delta_i, \Delta_i, \psi''_{j,i})\}_{j \neq i}$ to $\mathcal{Z}$, where $\{\delta_i, \Delta_i, \psi''_{j,i}\}_{i \in \boldsymbol{H}}$ are computed as prescribed.

2. Else, retrieve $\{\beta_{j,k}, \hat{\beta}_{j,k}\}_{j,k}$, and set

$$\begin{cases} \chi_b = k_b x_b + \sum_{j \neq b}(k_b x_j - \hat{\beta}_{j,b}) + (k_j x_b - \hat{\alpha}_{j,b}) \\ \delta_b = k_b \gamma_b + \sum_{j \neq b}(k_b \gamma_j - \beta_{j,b}) + (k_j \gamma_b - \alpha_{j,b}) \end{cases} .$$

Invoke ZK-simulator $\psi''_{j,b} \leftarrow \mathcal{S}^{\mathsf{log}}_j(\Delta_b, \Gamma, K_b, \ldots)$, for $j \neq b$.

Hand over $\{(sid, i, \delta_i, \Delta_i, \psi''_{j,i})\}_{j \neq i}$ to $\mathcal{Z}$, where $\{\delta_i, \Delta_i, \psi''_{j,i}\}_{i \in \boldsymbol{H}}$ are computed as prescribed.

Output. Upon receiving all $(sid, j, \delta_j, \Delta_j, \psi''_{i,j})$ for $j \neq i$, do:

1. If $\mathsf{aux} = (R, \delta)$, output (sid, η^0, η^1) and $(sid, \ell, R, k_i, \chi_i)_{i \in \boldsymbol{H}}$.
2. Else, set $R = \Gamma^{(\sum_j \delta_j)^{-1}}$ and output $(sid, \ell, R, k_i, \chi_i)_{i \notin \boldsymbol{C}}$.

References

[1] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. *J. Cryptology*, 17(4):297–319, 2004. doi: 10.1007/s00145-004-0314-9. URL https://doi.org/10.1007/s00145-004-0314-9.

[2] D. Boneh, R. Gennaro, and S. Goldfeder. Using level-1 homomorphic encryption to improve threshold DSA signatures for bitcoin wallet security. In *Progress in Cryptology - LATINCRYPT 2017 - 5th International Conference on Cryptology and Information Security in Latin America, Havana, Cuba, September 20-22, 2017, Revised Selected Papers*, pages 352–377, 2017.

[3] F. Boudot. Efficient proofs that a committed number lies in an interval. In B. Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, pages 431–444, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. ISBN 978-3-540-45539-4.

[4] E. F. Brickell, D. Chaum, I. B. Damgård, and J. van de Graaf. Gradual and verifiable release of a secret (extended abstract). In C. Pomerance, editor, *Advances in Cryptology — CRYPTO '87*, pages 156–166, Berlin, Heidelberg, 1988. Springer Berlin Heidelberg. ISBN 978-3-540-48184-3.

[5] D. R. L. Brown. The exact security of ecdsa. Technical report, Advances in Elliptic Curve Cryptography, 2000.

[6] D. R. L. Brown. Generic groups, collision resistance, and ECDSA. *Des. Codes Cryptogr.*, 35(1):119–152, 2005. URL http://www.springerlink.com/index/10.1007/s10623-003-6154-z.

[7] J. Camenisch and M. Michels. Proving in zero-knowledge that a number is the product of two safe primes. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, pages 107–122, 1999. doi: 10.1007/3-540-48910-X_8. URL https://doi.org/10.1007/3-540-48910-X_8.

[8] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, pages 126–144, 2003. doi: 10.1007/978-3-540-45146-4_8. URL https://doi.org/10.1007/978-3-540-45146-4_8.

[9] J. Camenisch, M. Drijvers, T. Gagliardoni, A. Lehmann, and G. Neven. The wonderful world of global random oracles. In J. B. Nielsen and V. Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018*, pages 280–312, Cham, 2018. Springer International Publishing. ISBN 978-3-319-78381-9.

[10] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145, 2001.

[11] R. Canetti. Universally composable signature, certification, and authentication. In *Proceedings. 17th IEEE Computer Security Foundations Workshop, 2004.*, pages 219–233, 2004.

[12] R. Canetti and S. Goldwasser. An efficient *Threshold* public key cryptosystem secure against adaptive chosen ciphertext attack. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, pages 90–106, 1999.

[13] R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Adaptive security for threshold cryptosystems. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 98–115, 1999.

[14] R. Canetti, A. Jain, and A. Scafuro. Practical UC security with a global random oracle. In G. Ahn, M. Yung, and N. Li, editors, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 597–608. ACM, 2014. doi: 10.1145/2660267.2660374. URL https://doi.org/10.1145/2660267.2660374.

[15] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker. Two-party ECDSA from hash proof systems and efficient instantiations. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*, pages 191–221, 2019.

[16] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker. Bandwidth-efficient threshold EC-DSA. *IACR Cryptology ePrint Archive*, 2020:84, 2020. URL https://eprint.iacr.org/2020/084.

[17] I. Damgård and M. Koprowski. Practical threshold RSA signatures without a trusted dealer. In *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, pages 152–165, 2001.

[18] Y. Desmedt. Society and group oriented cryptography: A new concept. In *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, pages 120–127, 1987. doi: 10.1007/3-540-48184-2_8. URL https://doi.org/10.1007/3-540-48184-2_8.

[19] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, pages 307–315, 1989. doi: 10.1007/0-387-34805-0_28. URL https://doi.org/10.1007/0-387-34805-0_28.

[20] J. Doerner, Y. Kondi, E. Lee, and A. shelat. Secure two-party threshold ecdsa from ecdsa assumptions. *2018 IEEE Symposium on Security and Privacy (SP)*, 2018.

[21] J. Doerner, Y. Kondi, E. Lee, and A. Shelat. Threshold ECDSA from ECDSA assumptions: The multiparty case. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 1051–1066, 2019. doi: 10.1109/SP.2019.00024. URL https://doi.org/10.1109/SP.2019.00024.

[22] M. Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, pages 152–168, 2005. doi: 10.1007/11535218_10. URL https://doi.org/10.1007/11535218_10.

[23] E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In B. S. Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, pages 16–30, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg. ISBN 978-3-540-69528-8.

[24] R. Gennaro and S. Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 1179–1194, 2018. doi: 10.1145/3243734.3243859. URL https://doi.org/10.1145/3243734.3243859.

[25] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. *Inf. Comput.*, 164 (1):54–84, 2001. doi: 10.1006/inco.2000.2881. URL https://doi.org/10.1006/inco.2000.2881.

[26] R. Gennaro, S. Goldfeder, and A. Narayanan. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In *Applied Cryptography and Network Security - 14th International Conference, ACNS 2016, Guildford, UK, June 19-22, 2016. Proceedings*, pages 156–174, 2016.

[27] S. Goldberg, L. Reyzin, O. Sagga, and F. Baldimtsi. Efficient noninteractive certification of RSA moduli and beyond. In *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part III*, pages 700–727, 2019.

[28] S. Goldwasser and Y. Lindell. Secure multi-party computation without agreement. *J. Cryptology*, 18(3):247–287, 2005. doi: 10.1007/s00145-005-0319-z. URL https://doi.org/10.1007/s00145-005-0319-z.

[29] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *Advances in Cryptology - CRYPTO '95, 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 27-31, 1995, Proceedings*, pages 339–352, 1995.

[30] S. Jarecki and J. Olsen. Proactive RSA with non-interactive signing. In *Financial Cryptography and Data Security, 12th International Conference, FC 2008, Cozumel, Mexico, January 28-31, 2008, Revised Selected Papers*, pages 215–230, 2008.

[31] D. Kravitz. Digital signature algorithm. US Patent 5231668A, 1993.

[32] Y. Lindell. Fast secure two-party ECDSA signing. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, pages 613–644, 2017. doi: 10.1007/978-3-319-63715-0_21. URL https://doi.org/10.1007/978-3-319-63715-0_21.

[33] Y. Lindell, A. Nof, and S. Ranellucci. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. *IACR Cryptology ePrint Archive*, 2018:987, 2018. URL https://eprint.iacr.org/2018/987.

[34] P. D. MacKenzie and M. K. Reiter. Two-party generation of DSA signatures. *Int. J. Inf. Sec.*, 2(3-4):218–239, 2004. doi: 10.1007/s10207-004-0041-0. URL https://doi.org/10.1007/s10207-004-0041-0.

[35] National Institute of Standards and Technology. Digital signature standard (dss). Federal Information Processing Publication 186-4, 2013. URL https://doi.org/10.6028/NIST.FIPS.186-4.

[36] R. Ostrovsky and M. Yung. How to withstand mobile virus attacks (extended abstract). In *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 19-21, 1991*, pages 51–59, 1991. doi: 10.1145/112600.112605. URL https://doi.org/10.1145/112600.112605.

[37] C. Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991. doi: 10.1007/BF00196725. URL https://doi.org/10.1007/BF00196725.

[38] V. Shoup. Practical threshold signatures. In *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, pages 207–220, 2000.

[39] V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *J. Cryptology*, 15(2):75–96, 2002. doi: 10.1007/s00145-001-0020-9. URL https://doi.org/10.1007/s00145-001-0020-9.

[40] J. van de Graaf and R. Peralta. A simple and secure way to show the validity of your public key. In *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, pages 128–134, 1987.

A Overview of the UC Model

We present a brief overview of the Universal Composability framework. We refer the reader to for further details. In the spirit of the classic simulation-based paradigm, the UC-framework is defined via a real and ideal process.

Real Process. The real process involves a set of n parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ executing a protocol π, an adversary $\mathcal{A}$ corrupting a subset of parties according to the adversarial model, and an environment $\mathcal{Z}$ given some initial string-advice z. All entities are modelled as efficient machines initialized on a security parameter κ. The environment $\mathcal{Z}$ activates the parties, chooses their input and reads their output, and communicates with $\mathcal{A}$ in an arbitrary fashion. The real process terminates when the environment outputs a bit. Let $\mathrm{REAL}_{\pi,\mathcal{A}}^{\mathcal{Z}(z)}(1^\kappa)$ denote the environment's output in the above process.

Ideal Process. Similarly to the real process, the ideal process involves a set of n parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$, an ideal-process adversary $\mathcal{S}$ corrupting a subset of parties according to the adversarial model, and environment $\mathcal{Z}$ given some initial string-advice z; all entities are modelled as efficient machines initialized on a security parameter κ. Contrary to the real-process, the ideal-process is further endowed with an ideal functionality $\mathcal{F}$ that interacts with all the entities in a predetermined fashion according to the ideal functionality's specifications. The environment $\mathcal{Z}$ activates the parties, chooses their input and reads their output, and communicates with $\mathcal{S}$ in an arbitrary fashion. The ideal process terminates when the environment outputs a bit. Let $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S}}^{\mathcal{Z}(z)}(1^\kappa)$ denote the environment's output in the above process.

Definition A.1. We say that π UC-realizes $\mathcal{F}$ if for every real adversary $\mathcal{A}$, there exists an ideal adversary $\mathcal{S}$ such that for every environment $\mathcal{Z}$ it holds that

$$\{\mathrm{REAL}_{\pi,\mathcal{A}}^{\mathcal{Z}(z)}(1^\kappa)\}_{z\in\{0,1\}^*,\kappa\in\mathbb{N}} \stackrel{c}{\equiv} \{\mathrm{IDEAL}_{\mathcal{F},\mathcal{S}}^{\mathcal{Z}(z)}(1^\kappa)\}_{z\in\{0,1\}^*,\kappa\in\mathbb{N}}.$$

Adversarial Model. We assume that the adversary is fully malicious and may deviate from the protocol arbitrarily. Furthermore, we allow the adversary to corrupt parties dynamically in-between phases as the protocol evolves, and to "decorrupt" some parties according to the specifications of the ideal functionality. We stress that this adversarial model is more versatile than the standard "static" variant where the parties are corrupted only at the beginning of the protocol and forever thereafter.

Communication Model. Following the standard definition, we assume that the parties are connected via pairwise authenticated channels. Furthermore, we assume the existence of a broadcast channel.

B More Sigma Protocols

B.1 Schnorr PoK (Π^{sch})

Figure 15 is a Σ-protocol for $(X; x)$ in the relation R_{sch}.

FIGURE 15 (Schnorr PoK – Π^{sch})

- **Inputs:** Common input is $(\mathbb{G}, q, g, X)$ where $q = |\mathbb{G}|$ and g is generators of $\mathbb{G}$.
  The Prover has secret input x such that $g^x = X$.

1. Prover samples $\alpha \leftarrow \mathbb{F}_q$ and sends $A = g^\alpha$ to the verifier.
2. Verifier replies with $e \leftarrow \mathbb{F}_q$
3. Prover sends $z = \alpha + ex \mod q$ to the verifier.

- **Verification:** Verifier checks that $g^z = A \cdot X^e$.

Figure 15: Schnorr PoK – Π^{sch}

B.2 Group Element vs Paillier Encryption in Range ZK (Π^{log})

Figure 16 is a Σ-protocol for the relation R_{log}.

FIGURE 16 (Knowledge of Exponent vs Paillier Encryption – Π^{log})

- **Setup:** Auxiliary safe bi-prime $\hat{N}$ and Ring-Pedersen parameters $s, t \in \mathbb{Z}^*_{\hat{N}}$.
- **Inputs:** Common input is $(\mathbb{G}, q, N_0, C, X, g)$.
  The Prover has secret input (x, ρ) such that $x \in \pm 2^\ell$, and $C = (1 + N_0)^x \cdot \rho^{N_0} \mod N_0^2$ and $X = g^x \in \mathbb{G}$.

1. Prover samples

$$\alpha \leftarrow \pm 2^{\ell+\varepsilon} \text{ and } \begin{cases} \mu \leftarrow \pm 2^\ell \cdot \hat{N} \\ r \leftarrow \mathbb{Z}^*_N \\ \gamma \leftarrow \pm 2^{\ell+\varepsilon} \cdot \hat{N} \end{cases}, \text{ and computes } \begin{cases} S = s^x t^\mu \mod \hat{N} \\ A = (1 + N_0)^\alpha \cdot R_{N_0} \mod N_0^2 \\ Y = g^\alpha \in \mathbb{G} \\ D = s^\alpha t^\gamma \mod \hat{N} \end{cases},$$

   and sends (S, A, Y, D) to the verifier.
2. Verifier replies with $e \leftarrow \pm q$
3. Prover sends (z_1, z_2, z_3, w) to the verifier, where

$$\begin{cases} z_1 &= \alpha + ex \\ z_2 &= r \cdot \rho^e \mod N_0 \\ z_3 &= \gamma + e\mu \end{cases}.$$

- **Equality Checks**:

$$\begin{cases} (1 + N_0)^{z_1} \cdot z_2^{N_0} = A \cdot C^e \mod N_0^2 \\ g^{z_1} = Y \cdot X^e \in \mathbb{G} \\ s^{z_1} t^{z_3} = D \cdot S^e \mod \hat{N} \end{cases}$$

- **Range Check**:

$$z_1 \in \pm 2^{\ell+\varepsilon}$$

  The proof guarantees that $x \in \pm 2^{\ell+\varepsilon}$.

Figure 16: Knowledge of Exponent vs Paillier Encryption – Π^{log}

B.3 Paillier Operation with Paillier Commitment ZK-Proof ($\Pi^{\mathsf{aff\text{-}p}}$)

Figure 17 is a Σ-protocol for the relation $R_{\mathsf{aff\text{-}p}}$.

FIGURE 17 (Paillier Affine Operation with Paillier Commitment ZK-Proof – $\Pi^{\mathsf{aff\text{-}p}}$)

- **Setup:** Auxiliary safe bi-prime $\hat{N}$ and Ring-Pedersen parameters $s, t \in \mathbb{Z}^*_{\hat{N}}$.
- **Inputs:** Common input is $(N_0, N_1, D, C, d, X, Y)$ where $q = |\mathbb{G}|$ and g a generator $\mathbb{G}$.

  The Prover has secret input $(x, y, \rho, \rho_x, \rho_y)$ such that $x \in \pm 2^\ell$, $y \in \pm 2^{\ell'}$, $(1+N_1)^x \rho_x^{N_1} = X$ and $(1+N_1)^y \rho_y^{N_1} = Y$ and $D = C^x (1+N_0)^y \cdot \rho^{N_0} \mod N_0^2$.

1. Prover samples $\alpha \leftarrow \pm 2^{\ell+\varepsilon}$ and $\beta \leftarrow \pm 2^{\ell'+\varepsilon}$ and

$$\begin{cases} r \leftarrow \mathbb{Z}^*_{N_0}, \\ r_x, r_y \leftarrow \mathbb{Z}^*_{N_1}, \\ \gamma \leftarrow \pm 2^{\ell+\varepsilon} \cdot \hat{N},\ m \leftarrow \pm 2^\ell \cdot \hat{N} \\ \delta \leftarrow \pm 2^{\ell+\varepsilon} \cdot \hat{N},\ \mu \leftarrow \pm 2^\ell \cdot \hat{N} \end{cases} \quad \text{and computes} \quad \begin{cases} A = C^\alpha \cdot ((1+(N_0)^\beta \cdot R_{N_0}) \mod N_0^2 \\ B_x = (1+N_1)^\alpha r_x^{N_1},\ B_y = (1+N_1)^\beta r_y^{N_1} \mod N_1^2 \\ E = s^\alpha t^\gamma,\ S = s^x t^m \mod \hat{N} \\ F = s^\beta t^\delta,\ T = s^y t^\mu \mod \hat{N} \end{cases}$$

   and sends (S, T, A, B, E, F) to the verifier.
2. Verifier replies with $e \leftarrow \pm q$.
3. Prover Prover sends $(z_1, z_2, z_3, z_4, w, w_x, w_y)$ to the verifier where

$$\begin{cases} z_1 = \alpha + ex \\ z_2 = \beta + ey \\ z_3 = \gamma + em \\ z_4 = \delta + e\mu \\ w = r \cdot \rho^e \mod N_0 \\ w_x = r_x \cdot \rho_x^e \mod N_1 \\ w_y = r_y \cdot \rho_y^e \mod N_1 \end{cases}$$

- **Equality Checks**:

$$\begin{cases} C^{z_1}(1+N_0)^{z_2} w^{N_0} = A \cdot D^e \mod N_0^2 \\ (1+N_1)^{z_1} w_x^{N_1} = B_x \cdot X^e \mod N_1^2 \\ (1+N_1)^{z_2} w_y^{N_1} = B_y \cdot Y^e \mod N_1^2 \\ s^{z_1} t^{z_3} = E \cdot S^e \mod \hat{N} \\ s^{z_2} t^{z_4} = F \cdot T^e \mod \hat{N} \end{cases}$$

- **Range Check**:

$$\begin{cases} z_1 \in \pm 2^{\ell+\varepsilon} \\ z_2 \in \pm 2^{\ell'+\varepsilon} \end{cases}$$

  The proof guarantees that $x \in \pm 2^{\ell+\varepsilon}$ and $y \in \pm 2^{\ell'+\varepsilon}$.

Figure 17: Paillier Affine Operation with Paillier Commitment ZK-Proof – $\Pi^{\mathsf{aff\text{-}p}}$

C Complexity Estimation

We provide computation and communication costs analysis of our protocol's components. The following figures depict the costs for *each* of the n parties. To ensure 80-bit statistical security and 128-bit computational security, we chose $m = 80$ in the Paillier-Blum Modulus and Ring-Pedersen ZK-Proofs, and in all other ZK-Range-Proofs ℓ, ℓ', ε are respectively $1, 5, 2$ factor of the Elliptic Curve element bit-length (e.g. for the Bitcoin curve secp256k1, $\ell = 256, \ell' = 1280, \varepsilon = 512$).

For computation we denote $\mathbf{G}, \mathbf{N}, \mathbf{N}^2$ as exponentiations in the EC group $\mathbb{G}$ and rings $\mathbb{Z}_N, \mathbb{Z}_{N^2}$, respectively. Communication is counted by the amount of EC elements required to transmit (with $\mathbb{Z}_N$ and $\mathbb{Z}_{N^2}$ elements counted as respective $8, 16$ EC elements to achieve required security). Hash (random oracle) invocations are insignificant, so were omitted from computational costs, but were counted as a single EC element for communication (which is in line with practice).

Component	*Rounds*	*Computation*	*Communication*
Key Generation	3	$(2 + 2n)\mathbf{G}$	4
Aux Info. & Key Refresh	2	$(n + 2n^2)\mathbf{G} + (400 + 321n + 3n^2)\mathbf{N} + (n + 2n^2)\mathbf{N}^2$	$3865 + 16n + 55n^2$
Pre-Signing	3	$(4 + 9n)\mathbf{G} + 57n\mathbf{N} + (2 + 32n)\mathbf{N}^2$	$35 + 444n$
Signing	1	0	1

The above figures were computed using following costs analysis of each of our non-interactive ZK-Proofs

ZK-Proof	*Computation (Prover)*	*Computation (Verifier)*	*Communication*
Π^{sch}	$1\mathbf{G}$	$2\mathbf{G}$	2
Π^{enc}	$5\mathbf{N} + 1\mathbf{N}^2$	$3\mathbf{N} + 2\mathbf{N}^2$	54
Π^{log}	$1\mathbf{G} + 5\mathbf{N} + 1\mathbf{N}^2$	$2\mathbf{G} + 3\mathbf{N} + 2\mathbf{N}^2$	55
$\Pi^{\mathsf{aff\text{-}g}}$	$1\mathbf{G} + 10\mathbf{N} + 3\mathbf{N}^2$	$2\mathbf{G} + 6\mathbf{N} + 5\mathbf{N}^2$	112
$\Pi^{\mathsf{aff\text{-}p}}$	$11\mathbf{N} + 4\mathbf{N}^2$	$6\mathbf{N} + 7\mathbf{N}^2$	136
Π^{mod}	$160\mathbf{N}$	$80\mathbf{N}$	1280
Π^{prm}	$80\mathbf{N}$	$160\mathbf{N}$	1280

We present some concrete values for the Pre-Signing and Aux Info. & Key Refresh, totaling over all rounds (but not including communication time). These were taken for Bitcoin's EC secp256k1 (and corresponding parameters), running a C implementation on Ubuntu Desktop with an Intel Quad-Core i7-7600 CPU @ 2.80GHz – without any optimization.

n	AI&KR	Pre-Signing
2	2228	801
3	3032	1183
4	3896	1566
5	4820	1949
6	5804	2332
7	6848	2715
8	7952	3098
9	9116	3864

Computation, in milliseconds

n	AI&KR	Pre-Signing
2	133	30
3	143	45
4	156	59
5	172	73
6	192	88
7	216	102
8	243	116
9	274	131

Communication, in kilobytes

D Number Theory & Probability Facts

Fact D.1. *Suppose that* $\lambda^N = x^k \mod M$ *such that* $x \in \mathbb{Z}_M^*$. *Then* $\lambda \in \mathbb{Z}_M^*$.

Proof. There exists $y \in \mathbb{Z}_M^*$ such that $xy = 1 \mod M$. Therefore $\lambda \cdot (\lambda^{N-1} \cdot y^k) = \lambda^N \cdot y^k = x^k y^k = 1 \mod M$. □

Fact D.2. *Suppose that* $\lambda^N = x^k \mod M$, *where* k *and* N *are coprime and* $x \in \mathbb{Z}_M^*$. *Then, there exists* $y \in \mathbb{Z}_M^*$ *such that* $y^k = \lambda \mod M$.

Proof. Since k and N are comprime, there exists $u, v \in \mathbb{Z}$ such that $ku + Nv = 1$. Thus $\lambda^{ku+Nv} = \lambda$, and consequently $(\lambda^u \cdot x^v)^k = \lambda^{ku} \cdot (\lambda^N)^v = \lambda \mod M$. For the penultimate equality, we apply Fact D.1 and we remark that λ^u and x^v are well defined in $\mathbb{Z}_M^*$. □

Remark D.3. We stress that computing a k-th root of λ in $\mathbb{Z}_M^*$ can be done efficiently via repeated application of Euclid's extended algorithm and exponentiation modulo M, i.e. computing the Bézout coefficients (u, v), as well as $\lambda^u \mod M$ and $x^v \mod M$.

Fact D.4. *Let* $a, c \in \mathbb{Z}$ *such that* $c \nmid a$. *There exists a prime power* p^d *such that* $p^{d-1} \mid a$, $p^d \nmid a$ *and* $p^d \mid c$.

Proof. Any prime factor that divides c but not a will do (taking $d = 1$). If no such p exists, i.e. if every prime factor of c divides a, let $p_1, \ldots, p_n$ denote the prime factors of a, and write $a = \prod_{j=1}^n p_i^{d_i}$ and $c = \prod_{j=1}^n p_j^{d'_j}$ (maybe some $d'_j = 0$). If $d'_i \leq d_i$, for every i, then $c \mid a$. Therefore, there exists i such that $d'_i > d_i$, and thus $(p, d) = (p_i, d_i + 1)$ will do. □

Fact D.5. *Let* $N = pq$ *be the product of two odd primes and let* x, y *and* $z \in \mathbb{Z}_N^*$ *such that* $x^2 = y^2 = z \mod N$ *and* $x \neq y, -y \mod N$. *Then* $\gcd(x - y, N) \in \{p, q\}$.

Proof. Let u, v denote the Bézout coefficients of the extended Euclid's algorithm such that $up + vq = 1$ and notice that $\gcd(p, v) = \gcd(q, u) = 1$. By Chinese remainder theorem, since $x \neq y, -y \mod n$, it follows that $x - y = 2cuq \mod N$ or $x + y = 2cvp \mod N$ for unique element $c \in \mathbb{Z}_p^*$ or $c \in \mathbb{Z}_q^*$, respectively. In either case, the claim follows. □

Fact D.6. *Define i.i.d. random variables* $\boldsymbol{a}, \boldsymbol{b}$ *chosen uniformly at random from* $\pm R$, *and let* $\delta \in \pm K$. *It holds that* $\mathrm{SD}(\boldsymbol{a}, \delta + \boldsymbol{b}) \leq K/R$.

Fact D.7. *Let* N *be the product of exactly two arbitrary primes* p *and* q. *Let* $\boldsymbol{a} \leftarrow \mathbb{Z}_{\ell \cdot N}$ *and* $\boldsymbol{b} \leftarrow \mathbb{Z}_{\phi(N)}$. *It holds that* $\mathrm{SD}(\boldsymbol{a} \mod \phi(N), \boldsymbol{b}) \leq \frac{1}{\ell}$.

Proof. Let $Q = \lfloor \ell \cdot N / \phi(N) \rfloor$ observe that $\mathrm{SD}(\boldsymbol{a} \mod \phi(N), \boldsymbol{b}) \leq \Pr[\boldsymbol{a} \geq Q \cdot \phi(N)]$. Thus, $\Pr[\boldsymbol{a} \geq Q \cdot \phi(N)] \leq \Pr[\boldsymbol{a} \geq \ell \cdot N - \phi(N)] = \phi(N)/(\ell \cdot N) \leq \frac{1}{\ell}$. □

E Assumptions

Definition E.1 (Semantic Security)**.** We say that the encryption scheme $(\mathsf{gen}, \mathsf{enc}, \mathsf{dec})$ is semantically secure if there exists a negligible function $\nu(\cdot)$ such that for every $\mathcal{A}$ it holds that $\Pr[\mathsf{PaillierSec}(\mathcal{A}, 1^\kappa) = 1] \leq 1/2 + \nu(\kappa)$.

Definition E.2 (Existential Unforgeability)**.** We say that a signature scheme $(\mathsf{gen}, \mathsf{sign}, \mathsf{vrfy})$ is existentially unforgeable if there exists a negligible function $\nu(\cdot)$ such that for every $\mathcal{A}$ and every $n \in \mathsf{poly}$ it holds that $\Pr[\mathsf{ExUnf}(\mathcal{A}, n, 1^\kappa) = 1] \leq \nu(\kappa)$.

Definition E.3 (Strong-RSA)**.** We say that strong-RSA is hard if there exists a negligible function $\nu(\cdot)$ such that for every $\mathcal{A}$ it holds that $\Pr[\mathsf{sRSA}(\mathcal{A}, 1^\kappa) = 1] \leq \nu(\kappa)$.

FIGURE 18 (Semantic Security Experiment $\mathsf{PaillierSec}(\mathcal{A}, 1^\kappa)$)

1. Generate a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{gen}(1^\kappa)$
2. $\mathcal{A}$ chooses m_0, $m_1 \in M$ on input $(1^\kappa, \mathsf{pk})$.
3. Compute $c = \mathsf{enc}_{\mathsf{pk}}(m_b)$ for $b \leftarrow \{0, 1\}$.
4. $\mathcal{A}$ outputs b' on input $(1^\kappa, \mathsf{pk}, m_0, m_1, c)$.

- **Output:** $\mathsf{PaillierSec}(\mathcal{A}, 1^\kappa) = 1$ if $b = b'$ and 0 otherwise.

Figure 18: Semantic Security Experiment $\mathsf{PaillierSec}(\mathcal{A}, 1^\kappa)$

FIGURE 19 (Existential Unforgeability Experiment $\mathsf{ExUnf}(\mathcal{A}, \mathcal{H}^{\mathsf{g}}, n, 1^\kappa)$)

1. Generate a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{gen}(1^\kappa)$ and let $(m_0, \sigma_0) = (\emptyset, \emptyset)$.
2. For $i = 1 \ldots n(\kappa)$
   - Choose $m_i \leftarrow \mathcal{A}^{\mathcal{H}^{\mathsf{g}}}(1^\kappa, \mathsf{pk}, m_0, \sigma_0, \ldots, m_{i-1}, \sigma_{i-1})$
   - Compute $\sigma_i = \mathsf{sign}_{\mathsf{pk}}(m_i)$.
3. $\mathcal{A}^{\mathcal{H}^{\mathsf{g}}}$ outputs (m, σ) on input $(1^\kappa, \mathsf{pk}, m_0, \sigma_0, \ldots, m_{n(\kappa)}, \sigma_{n(\kappa)})$.

- **Output:** $\mathsf{ExUnf}(\mathcal{A}, \mathcal{H}^{\mathsf{g}}, n, 1^\kappa) = 1$ if $\mathsf{vrfy}_{\mathsf{pk}}(m, \sigma) = 1$ and $m \notin \{m_1, \ldots, m_{n(\kappa)}\}$ and 0 otherwise.

Figure 19: Existential Unforgeability Experiment $\mathsf{ExUnf}(\mathcal{A}, \mathcal{H}^{\mathsf{g}}, n, 1^\kappa)$

FIGURE 20 (Strong-RSA Experiment $\mathsf{sRSA}(\mathcal{A}, 1^\kappa)$)

1. Generate an RSA modulus $N \leftarrow \mathcal{N}(1^\kappa)$.
2. Sample $c \leftarrow \mathbb{Z}_N^*$.
3. $\mathcal{A}$ outputs (m, e) on input $(1^\kappa, N, m)$.

- **Output:** $\mathsf{sRSA}(\mathcal{A}, 1^\kappa) = 1$ if $e > 1$ and $m^e = c \mod N$, and 0 otherwise.

Figure 20: Strong-RSA Experiment $\mathsf{sRSA}(\mathcal{A}, 1^\kappa)$

E.1 Enhanced Existential Unforgeability of ECDSA

E.1.1 $O(1)$-Enhanced Forgeries

Lemma E.4. *If ECDSA is existentially unforgeable, then there exists a negligible function ν such that for any PPTM $\mathcal{A}$, for every $T \in \mathsf{poly}(\kappa)$ and $S \in O(1)$ it holds that* $\Pr[\mathsf{EnhancedECDSA}(\mathcal{A}, S, T, \kappa) = 1] \in \nu(\kappa)$.

FIGURE 21 (ECDSA Multi-Enhanced Experiment $\mathsf{EnhancedECDSA}(\mathcal{A}, \mathcal{H}^{\mathsf{g}}, S, T, 1^\kappa)$)

0. Choose a group-order-generator tuple $(\mathbb{G}, q, g) \leftarrow \mathsf{gen}(1^\kappa)$.
1. Generate a key-pair $(x \leftarrow \mathbb{F}_q, X = g^x)$ and let $(\boldsymbol{R}_0, \boldsymbol{m}_0, \boldsymbol{\sigma}_0) = (\emptyset, \emptyset, \emptyset)$.
2. For $i = 1 \ldots T$
   - Sample $\boldsymbol{R}_i = \{R_{i,j} = g^{k_{i,j}^{-1}} \leftarrow \mathbb{G}\}_{j \le S}$.
   - For $j = 1 \ldots S$
     (a) Choose $m_{i,j} \leftarrow \mathcal{A}^{\mathcal{H}^{\mathsf{g}}}(\mathbb{G}, g, \boldsymbol{R}_0, \boldsymbol{m}_0, \boldsymbol{\sigma}_0, \ldots, \boldsymbol{R}_{i-1}, \boldsymbol{m}_{i-1}, \boldsymbol{\sigma}_{i-1}, \boldsymbol{R}_i, m_{i,<j}, \sigma_{i,<j})$
     (b) Compute $\sigma_{i,j} = \mathsf{sign}_x(m_{i,j}; k_{i,j})$.
     Set $\boldsymbol{m}_i = \{m_{i,j}\}_j$ and $\boldsymbol{\sigma}_i = \{\sigma_{i,j}\}_j$.
3. $\mathcal{A}^{\mathcal{H}^{\mathsf{g}}}$ outputs (m, σ) on input $(\mathbb{G}, g, \boldsymbol{R}_0, \boldsymbol{m}_0, \boldsymbol{\sigma}_0, \ldots, \boldsymbol{R}_T, \boldsymbol{m}_T, \boldsymbol{\sigma}_T)$.

- **Output:** $\mathsf{EnhancedECDSA}(\mathcal{A}, \mathcal{H}^{\mathsf{g}}, T, \mathbb{G}) = 1$ if $\mathsf{vrfy}_X(m, \sigma) = 1$ and $m \notin \{m_{i,j}\}_{i,j}$ and 0 otherwise.

Figure 21: ECDSA Multi-Enhanced Experiment $\mathsf{EnhancedECDSA}(\mathcal{A}, \mathcal{H}^{\mathsf{g}}, S, T, 1^\kappa)$

Proof. Let $Q \in \mathsf{poly}$ denote the number of oracle queries that the adversary makes in between each signature query. We show that any adversary that wins the experiment above with noticeable probability p yields an efficient adversary that forges signatures with the same probability in the (plain) ECDSA experiment and complexity most $T \cdot Q^S log(Q) \in \mathsf{poly}$ queries. Define process $\mathcal{R}$ with black-box access to $\mathcal{A}$ as follows: choose Q messages uniformly at random denoted $\{m'_i\}_{i \in [Q]}$. Then choose $I^* \subset [Q]$ of size S uniformly at random and invoke the (plain) ECDSA oracle on m'_{i^*}, for every $i^* \in I^*$. Write $(R_{i^*}, M_{i^*} = \mathcal{H}^{\mathsf{g}}(m'_{i^*}), \sigma_{i^*})$ for the signature. Next do:

1. Hand over $\{R_{i^*}\}_{i^* \in I^*}$ to $\mathcal{A}$
2. For $i = 1 \ldots Q$, each time $\mathcal{A}$ queries the oracle on m_i, hand over $(\mathtt{answer}, M_i = \mathcal{H}^{\mathsf{g}}(m'_i))$.
3. When $\mathcal{A}$ queries the ECDSA oracle on m_{j^*}, do
   - If $j^* \neq i^*$ rewind the adversary and repeat.
   - Else hand over σ.

Observe that $\Pr[\forall i^*,\ i^* = j^*] = \frac{1}{\binom{Q}{S} \cdot S!} \in O(1/Q^S)$ and that the reduction will guess every j^* with probability close to 1 after $Q^S \cdot \log(Q)$ tries.

□

E.1.2 Multi-Enhanced Forgeries: Preliminaries

Brief overview of the Generic Group Model. Let $(\mathbb{G}, q, g)$ denote a group-order-generator tuple and let $\boldsymbol{G} \subset \{0,1\}^*$ denote an arbitrary set of size q. The generic group model is defined via a random bijective map $\mu : \mathbb{G} \to \boldsymbol{G}$ and a group-oracle $\mathcal{O} : \boldsymbol{G} \times \boldsymbol{G} \to \boldsymbol{G}$ such that $\mu(gh) = \mathcal{O}(\mu(g), \mu(h))$, for every $g, h \in \mathbb{G}$. In group-theoretic jargon, $(\boldsymbol{G}, *)$ is isomorphic to $(\mathbb{G}, \cdot)$ via the group-isomorphism μ, letting $* : \boldsymbol{G} \times \boldsymbol{G} \to \boldsymbol{G}$ such that $G * H = \mathcal{O}(G, H)$.

EC-specific abstraction. We further assume that there exists an efficient 2-to-1 map $\tau : \boldsymbol{G} \to \mathbb{F}_q$ such that $\tau(H) = \tau(H^{-1})$. We further assume that this map is efficiently invertible $\tau^{-1} : \mathbb{F}_q \to \{\{G, H\} \text{ s.t. } G, H \in \boldsymbol{G}\} \cup \{\bot\}$ such that

$$\tau^{-1} : x \mapsto \begin{cases} \{H, H^{-1}\} & \text{if } \exists H \text{ s.t. } \tau(H) = x \\ \bot & \text{otherwise} \end{cases}.$$

Notation E.5. Define π such that $\pi(X) = \emptyset$ and $\pi(X_1, \ldots, X_\ell) = (X_1, X_2, X_1X_2) \parallel \pi(X_1X_2, X_3 \ldots, X_\ell)$, for every $X, X_1, \ldots X_\ell \in \boldsymbol{G}$. Furthermore, for $X \in \boldsymbol{G}$ and $k \in \mathbb{F}_q$ let $(k_i)_{i \leq q_0}$ denote the binary representation of k and define

$$(X^k) = \begin{cases} (X, X, X^2, \ldots, X^{k/2}, X^{k/2}, X^k) & \text{if } k \text{ is a power of 2} \\ (\mathsf{id}_{\boldsymbol{G}}, X, X) \parallel (X^{k_1 \cdot 2}) \parallel \ldots \parallel (X^{k_{q_0} \cdot 2^{q_0}}) \parallel \pi(X^{k_0}, \ldots, X^{k_{q_0} 2^{q_0}}) & \text{otherwise} \end{cases},$$

where $q_0 = \lfloor \log q \rfloor$.

FIGURE 22 (ECDSA Experiment in Generic Group w/ Enhanced Signing Oracle)

- Group Oracle $\mathcal{O}$:
  - On input (X, Y), return $Z = X * Y$.
    Set $\boldsymbol{Q} = \boldsymbol{Q} \parallel (X, Y, Z)$.
- Signing oracle $\mathcal{S}^{\mathcal{O}}$:
  - On input `pubkey`, sample $G \leftarrow \boldsymbol{G}$ and $x \leftarrow \mathbb{F}_q$ and return $(G, H = G^x)$.
  - On input `pnt-request`, sample $k \leftarrow \mathbb{F}_q$, return $R = G^{k^{-1}}$, add R to $\boldsymbol{R}$ and record (R, k).
  - On input $(\texttt{sign}, \text{msg}, R)$, if $R \in \boldsymbol{R}$ retrieve (R, k) and do:
    1. Return $\sigma = k(m + rx)$, for $r = \tau(R)$ and $m = \mathcal{H}^{\mathsf{g}}(\text{msg})$.
    2. Set $\boldsymbol{Q} = \boldsymbol{Q} \parallel (G^{m/\sigma}) \parallel (H^{r/\sigma}) \parallel (G^{m/\sigma}, H^{r/\sigma}, R)$.
    3. Remove R from $\boldsymbol{R}$ and add (R, m, σ) to $\boldsymbol{S}$.

Figure 22: ECDSA Experiment in Generic Group w/ Enhanced Signing Oracle

Let $\mathcal{A}$ denote an algorithm interacting with $\mathcal{O}, \mathcal{S}^{\mathcal{O}}$ in the experiment described in Figure 22. Consider the tuple of all oracle calls $\boldsymbol{Q} = (Q_1, \ldots, Q_{3t}) = (X_1, Y_1, Z_1, \ldots, X_t, Y_t, Z_t)$, where each pair (X_i, Y_i) denotes the input to $\mathcal{O}$ and Z_i denotes the output.

Definition E.6. We say that $Q_i \in \{X_j, Y_j\}$ is *independent* if $(Q_i, \ldots) \notin \boldsymbol{S}$ *and* $Q_i \notin \{Q_k, Q_k^{-1}\}$, for every $k < i$.

Lemma E.7 (Brown [5, 6])**.** *The following holds with all but negligible probability for every efficient algorithm $\mathcal{A}$ interacting with $\mathcal{O}$. Let $B_1 \ldots B_\ell$ denote the independent elements of $\boldsymbol{Q}$ and let $Q \in \boldsymbol{Q}$. Suppose that $\mathcal{A}$ outputs two sequences $(\alpha_1, \ldots, \alpha_\ell)$ and $(\alpha'_1, \ldots, \alpha'_\ell)$ such that*

$$Q = \prod_{k \leq \ell} B_k^{\alpha_k} = \prod_{k \leq \ell} B_k^{\alpha'_k}.$$

Then, with probability $1 - 1/\mathsf{poly}(q)$ it holds that $\alpha_i = \alpha'_i \mod q$, for every $i \in [\ell]$. Furthermore, if $Q = Z_j$, then $\alpha_1, \ldots, \alpha_\ell$ is determined by $(X_i, Y_i, Z_i)_{i<j}$ and $\boldsymbol{S}$.

E.1.3 Multi-Enhanced Forgeries: Proof

Theorem E.8. *Let $\mathcal{A}$ be an algorithm in the generic group experiment with enhanced signing oracle making ℓ queries to the random oracle. If $\mathcal{A}$ outputs a forgery with probability α, then there exists $\mathcal{B}$ making at most ℓ queries to the random oracle such that*

$$\Pr_{e \leftarrow \mathbb{F}_q}[(x, y) \leftarrow \mathcal{B}(e) \ \ s.t. \ \ \mathcal{H}^{\mathsf{g}}(x)/\mathcal{H}^{\mathsf{g}}(y) = e] \geq \alpha/t - 1/\mathsf{poly}(q),$$

where t denotes the number of calls to the group operation.

FIGURE 23 (Reduction in Generic Group w/ Enhanced Signing Oracle)

- Group operations:
  - On input (X, Y), do:
    1. If $\phi(Z) = \phi(X * Y)$ for some $Z \in \boldsymbol{Q}$, return Z.
    2. Else If $\phi(X * Y) = G^{\alpha} H^{\beta}$ for $\alpha, \beta \neq 0$ do:
       (a) Sample y and $e \leftarrow \mathbb{F}_q$ and set $w = e \cdot \mathcal{H}^{\mathsf{g}}(y)$ and $Z \leftarrow \tau^{-1}(\alpha^{-1} w \beta)$.
       If $\tau^{-1}(\alpha^{-1} w \beta) = \perp$, repeat the above step.
       (b) Return Z.
    3. Else if $\phi(X * Y) = G^{\alpha} H^{\beta} \prod_{i \leq \ell} R_i^{\gamma_i}$ for $R_i \in \boldsymbol{R}$ and $\gamma_i \neq 0$ do:
       (a) Choose $i \leftarrow [\ell]$ and $e \leftarrow \mathbb{F}_q$ and set $Z \leftarrow \tau^{-1}(e \cdot r_i)$, for $r_i = \tau(R_i)$.
       If $\tau^{-1}(e r_i) = \perp$, repeat the above step.
       (b) Return Z.
    4. Else return $Z \leftarrow \boldsymbol{G}$.

    Set $\boldsymbol{Q} = \boldsymbol{Q} \parallel (X, Y, Z)$.
- Signing operations:
  - On input `pubkey`, return $(G, H) \leftarrow \boldsymbol{G}^2$.
  - On input `pnt-request`, return $R \leftarrow \boldsymbol{G}$, and add R to $\boldsymbol{R}$.
  - On input (`sign`, msg, R), if $R \in \boldsymbol{R}$ set $m = \mathcal{H}^{\mathsf{g}}(\text{msg})$ and $r = \tau(R)$, and do:
    1. Choose $Z \leftarrow \boldsymbol{Q}$ such that $\phi(Z) = G^{\alpha} H^{\beta} R^{\gamma}$, for $\gamma \neq 0$ and $\beta m - r\alpha \neq 0$.
       (a) Sample y and $e \leftarrow \mathbb{F}_q$ and set $w = e \cdot \mathcal{H}^{\mathsf{g}}(y)$ and $\sigma = \gamma(w r \zeta^{-1} - m) \cdot (\alpha - w\zeta^{-1}\beta)^{-1}$, for $\zeta = \tau(Z)$.
       (b) If no such Z exists set $\sigma \leftarrow \mathbb{F}_q$.
    2. Program $G^{m/\sigma}$, $H^{r/\sigma}$ and $G^{m/\sigma} * H^{r/\sigma}$ such that $G^{m/\sigma} * H^{r/\sigma} = R$ using group rules.
    3. Return σ and remove R from $\boldsymbol{R}$, and add (R, m, σ) to $\boldsymbol{S}$.

    Set $\boldsymbol{Q} = \boldsymbol{Q} \parallel (G^{m/\sigma}) \parallel (H^{r/\sigma}) \parallel (G^{m/\sigma}, H^{r/\sigma}, R)$.

Figure 23: Reduction in Generic Group w/ Enhanced Signing Oracle

The above theorem follows from the claim below by straightforward averaging argument.

Claim E.9. *Let $\mathcal{A}$ be an algorithm in the generic group experiment with enhanced signing oracle making ℓ queries to the random oracle. If $\mathcal{A}$ outputs a forgery with probability α, then there exists $\mathcal{B}$ making at most ℓ queries to the random oracle such that*

$$\Pr_{e_1,\ldots,e_t \leftarrow \mathbb{F}_q}[(x, y) \leftarrow \mathcal{B}(e_1, \ldots, e_t) \ : \ \exists i \ \ s.t. \ \ \mathcal{H}^{\mathsf{g}}(x)/\mathcal{H}^{\mathsf{g}}(y) = e_i] \geq \alpha - 1/\mathsf{poly}(q)$$

where t denotes the number of calls to the group operation.

Proof. Using the notation above, for a tuple of query calls $\boldsymbol{Q} = (Q_1 \ldots)$ and signed points $\boldsymbol{S}$, let $\phi : \boldsymbol{G} \to (\mathbb{F}_q)^*$ denote the function that maps group-elements to their representation with respect to the independent points of $\boldsymbol{Q}$. Namely $\phi(Q_i) = \prod_k B_k^{\alpha_k}$ as (uniquely) determined by $(Q_j)_{j<i}$ and $\boldsymbol{S}$. To conclude, consider the reduction from Figure 23, and the claim follows by observing that if $\gamma m \neq 0$ and $\beta m - r\alpha \neq 0$, then $\sigma \mapsto \zeta(\beta + \gamma r \sigma^{-1})^{-1}(\alpha + \gamma m \sigma^{-1})$ is injective. ☐

F Pre-Signing w/o Point-Reveal

In this section we present a presigning phase that does not reveal the point a the end of the computation. The protocol is formally described in Figure 24. Next, we give a high-level overview.

1. Each party $\mathcal{P}_i$ generates local shares k_i and γ_i and they compute Paillier encryptions K_i (to k_i) and G_i (to γ_i) and $D_{i,i}$ (to $k_i\gamma_i$) , under $\mathcal{P}_i$'s key. Then, each party sends to everyone (multicasts) the three-tuple $(D_{i,i}, K_i, G_i)$ together with the following
   (a) ZK-Proof that the plaintext of K_i lies in range $\mathcal{I}_\varepsilon$.
   (b) ZK-Proof that the plaintext of $D_{i,i}$ is equal to the product of the plaintexts of K_i, G_i.
2. For each $j \neq i$, party $\mathcal{P}_i$ samples $(\beta_{i,j}, \beta'_{i,j}) \leftarrow \mathcal{J}^2$ and computes $D_{j,i} = \mathsf{enc}_j(\gamma_i \cdot k_j - \beta_{i,j})$ and $D'_{j,i} = \mathsf{enc}_j(x_i \cdot k_j - \beta'_{i,j})$ using the homomorphic properties of Paillier. Furthermore, P_i encrypts $F_{i,j} = \mathsf{enc}_i(\beta_{i,j})$ and $F'_{i,j} = \mathsf{enc}_i(\beta'_{i,j})$ under his own Paillier key. Then, $\mathcal{P}_i$ is instructed to send the tuple $(D_{j,i}, D'_{j,i}, F_{i,j}, F'_{i,j})$ to all parties, together with the following ZK-PoKs
   (a) Ciphertext $D_{j,i}$ was obtained as an affine-like opperation on K_j where the multiplicative coefficient is equal to the hidden value of G_i, and it lies in range $\mathcal{I}_\varepsilon$, and the additive coefficient is equal to hidden value of $F_{i,j}$, and lies in range $\mathcal{J}_\varepsilon$.
   (b) Ciphertext $D'_{j,i}$ was obtained as an affine operation on K_j where the multiplicative coefficient is equal to the exponent of X_i, and it lies in range $\mathcal{I}_\varepsilon$, and the additive coefficient is equal to hidden value of $F'_{i,j}$, and it lies in range $\mathcal{J}_\varepsilon$.

Each $\mathcal{P}_i$ decrypts (and reduces modulo q) $\alpha_{i,j} = \mathsf{dec}_i(D_{i,j})$ and $\alpha'_{i,j} = \mathsf{dec}_i(D'_{i,j})$ and sets $\chi_i = k_i x_i + \sum_{j\neq i} \alpha'_{i,j} + \beta'_{i,j} \mod q$ and $\delta_i = k_i\gamma_i + \sum_{j\neq i} \alpha_{i,j} + \beta_{i,j} \mod q$. Party $\mathcal{P}_i$ sets $\Gamma_i = g^{\gamma_i}$ and $\{D_j = \prod_k D_{j,k}\}_j$, and records the following ZK-Proof, denote ψ_i and $\hat{\psi}_i$ respectively.

1. The exponent base g of Γ_i is equal to the plaintext value of G_i.
2. δ_i is the plaintext value of D_i modulo q.

Each $\mathcal{P}_i$ stores $\Gamma_i, \{G_j, D_j\}_j, \psi_i, \hat{\psi}_i, \delta_i, k_i, \chi_i$.

Point-Reveal. Before they are able to compute their signature-shares, the parties need to reveal one of the curve-points they preprocessed in the pre-signing phase. Namely, on input $(\texttt{pt-rvl}, \ell, i)$, each $\mathcal{P}_i$ retrieves the tuple $(\ell, \Gamma_i, \{G_j, D_j, \psi_{j,i}\}_j, \psi_i, \delta_i, k, \chi)$, and sends $(\delta_i, \Gamma_i, \psi_i, \psi_{j,i})$ to all.

After receiving all the relevant messages from the other parties, if no inconsistencies were detected, compute $\delta = \sum_j \delta_j$ and $R = \prod_j (\Gamma_j)^{\delta^{-1}}$. Record (ℓ, R, k, χ) and erase the retrieved data.

FIGURE 24 (ECDSA Pre-Signing w/o Point-Reveal)

Recall that P_i's secret state contains x_i, p_i, q_i such that $X_i = g^{x_i}$ and $N_i = p_i q_i$.

Round 1.

On input $(\texttt{pre-sign}, sid, \ell, i)$ from $\mathcal{P}_i$, interpret $sid = (\ldots, \boldsymbol{X}, \boldsymbol{N}, \boldsymbol{s}, \boldsymbol{t})$, and do:

- Sample $k_i,\ \gamma_i \leftarrow \mathbb{F}_q$ and $\rho_i, \nu_i, \mu_i \leftarrow \mathbb{Z}^*_{N_i}$
- Set $G_i = \mathsf{enc}_i(\gamma_i; \nu_i)$, $K_i = \mathsf{enc}_i(k_i; \rho_i)$, $D_{i,i} = \mathsf{enc}_i(\gamma_i k_i; \mu_i)$
- Compute

$$\begin{cases} \psi^0_{j,i} = \mathcal{M}(\texttt{prove}, \Pi^{\mathsf{enc}}_j, sid_i, K_i, \mathcal{I}_\varepsilon) & \text{for } j \neq i \\ \psi^0_i = \mathcal{M}(\texttt{prove}, \Pi^{\mathsf{mul}}, sid_i, D_{i,i}, K_i, G_i) \end{cases}.$$

Broadcast (sid_i, K_i, G_i) and send $(sid_i, D_{i,i}, \psi^0_{j,i}, \psi^0_i)$ to each $\mathcal{P}_j \in \boldsymbol{P}$.

Round 2.

1. Upon receiving $(sid_j, K_j, G_j, D_{j,j}, \psi^0_{i,j}, \psi^0_i)$, do:
   - Compute $b^0_j = \mathcal{M}(\texttt{vrfy}, \Pi^{\mathsf{enc}}_i, sid_j, K_j, \mathcal{I}_\varepsilon, \psi^0_{i,j})$.
   - Compute $\hat{b}^0_j = \mathcal{M}(\texttt{vrfy}, \Pi^{\mathsf{mul}}, sid_j, D_{j,j}, K_j, G_j, \psi^0_i)$.

   If $(b^0_j, \hat{b}^0_j) \neq (1,1)$, report a complaint and halt.
2. When obtaining all tuples do:

   (a) For $j \neq i$, sample $r_{i,j}, s_{i,j}, \hat{r}_{i,j}, \hat{s}_{i,j} \leftarrow \mathbb{Z}_N$ and $\beta_{i,j}, \beta'_{i,j} \leftarrow \mathcal{J}$ and set

$$\begin{cases} D'_{j,i} = (K_j \odot x_i) \oplus \mathsf{enc}_j(-\beta'_{i,j}\ ,\ r_{i,j}) \\ F'_{i,j} = \mathsf{enc}_i(\beta'_{i,j}\ ,\ \hat{r}_{i,j}) \end{cases}, \begin{cases} D_{j,i} = (K_j \odot \gamma_i) \oplus \mathsf{enc}_j(-\beta_{i,j}\ ,\ s_{i,j}) \\ F_{i,j} = \mathsf{enc}_i(\beta_{i,j}\ ,\ \hat{s}_{i,j}) \end{cases}$$

   (b) Compute
   - $\psi'_{j,i} = \mathcal{M}(\texttt{prove}, \Pi^{\mathsf{aff}}_j, sid_i, D'_{j,i}, K_j, X_i, F'_{i,j}, \mathcal{I}_\varepsilon, \mathcal{J}_\varepsilon)$.
   - $\psi_{j,i,k} = \mathcal{M}(\texttt{prove}, \Pi^{\mathsf{aff}}_j, sid_i, D_{k,i}, K_k, G_i, F_{i,k}, \mathcal{I}_\varepsilon, \mathcal{J}_\varepsilon)$, for $k \neq i$.

   Send $(sid_i, D'_{j,i}, F'_{i,j}, \psi'_{j,i}, \{D_{k,i}, F_{i,k}, \psi_{j,i,k}\}_{k\neq i})$ to $\mathcal{P}_j$.

Output.

1. Upon receiving $(sid_j, D'_{i,j}, F'_{j,i}, \psi'_{j,i}, \{D_{k,j}, F_{j,k}, \psi_{i,j,k}\}_{k\neq j})$ from $\mathcal{P}_j$,

   Compute
   - $b'_j = \mathcal{M}(\texttt{vrfy}, \Pi^{\mathsf{aff}}_i, sid_j, D'_{i,j}, K_i, X_j, F'_{j,i}, \mathcal{I}_\varepsilon, \mathcal{J}_\varepsilon, \psi'_{i,j})$
   - $b_{j,k} = \mathcal{M}(\texttt{vrfy}, \Pi^{\mathsf{aff}}_i, sid_j, D_{k,j}, K_k, G_j, F_{j,k}, \mathcal{I}_\varepsilon, \mathcal{J}_\varepsilon, \psi_{i,j,k})$, for $k \neq j$.

   If $b'_j \neq 1$ or $(b_{j,k})_{k\neq j} \neq (1, \ldots, 1)$, report a complaint and halt.
2. When obtaining all the b's, do:

   (a) Compute
   - $\{\alpha_{i,j} = \mathsf{dec}_i(D_{i,j})\}_j$ and $\{\alpha'_{i,j} = \mathsf{dec}_i(D'_{i,j})\}_j$ and

$$\begin{cases} \chi_i = k_i x_i + \sum_{j\neq i}(\alpha'_{i,j} + \beta'_{i,j}) \mod q \\ \delta_i = k_i \gamma_i + \sum_{j \neq i}(\alpha_{i,j} + \beta_{i,j}) \mod q \end{cases}$$

   - Set $d_i = \prod_j D_{i,j}$ and $\Gamma_i = g^{\gamma_i}$ and do: for $j \neq i$ compute

$$\begin{cases} \psi_{j,i} = \mathcal{M}(\texttt{prove}, \Pi^{\mathsf{dec}}_j, sid_i, \delta_i, D_i, \mathcal{J}_{2\varepsilon}) \\ \hat{\psi}_{j,i} = \mathcal{M}(\texttt{prove}, \Pi^{\mathsf{log}}_j, sid_i, \Gamma_i, g, G_i) \end{cases}.$$

   (b) Set $\{D_j = \prod_k D_{j,k}\}_j$ and output $(sid, \ell, \Gamma_i, \{G_j, D_j, \psi_{j,i}, \hat{\psi}_i\}_j, \delta_i, k_i, \chi_i)$.
   Erase all items in memory except for the stored state.

Errors. Upon receiving a complaint from any other $\mathcal{P}_j \in \boldsymbol{P}$, report a complaint and halt.

Stored State. Store $\boldsymbol{X}$, $\boldsymbol{N}$, $\boldsymbol{s}$, $\boldsymbol{t}$ and (x_i, p_i, q_i).

Figure 24a: ECDSA Pre-Signing w/o Point-Reveal

FIGURE 25 (ECDSA Point-Reveal)

Round 1.

On input $(\texttt{pt-rvl}, sid, \ell, i)$,

(a) If there is record of $(sid, \ell, \Gamma_i, \{G_j, D_j, \psi_{j,i}, \hat{\psi}_{j,i}\}_j, \delta_i, k, \chi)$,
Send the tuple $(sid_i, \delta_i, \Gamma_i, \psi_{j,i}, \hat{\psi}_{j,i})$ to $\mathcal{P}_j$.

(b) Else, do nothing.

Output.

1. Upon receiving $(sid_j, \delta_j, \Gamma_j, \psi_{j,i}, \hat{\psi}_{j,i})$ from $\mathcal{P}_j$ compute
   - $b_j = \mathcal{M}(\texttt{vrfy}, \Pi_i^{\mathsf{dec}}, sid_j, \delta_j, D_j, \psi_{i,j})$
   - $\hat{b}_j = \mathcal{M}(\texttt{vrfy}, \Pi_i^{\mathsf{log}}, sid_j, \Gamma_j, g, G_j, \hat{\psi}_{i,j})$

   If $(b_j, \hat{b}_j) \neq (1, 1)$, report a complaint and halt.
2. When obtaining all b_j's,
   - Set $R = (\prod_j \Gamma_j)^{(\sum_j \delta_j)^{-1}}$.
   - Record the tuple (sid, ℓ, R, k, χ)
   - Erase the tuple $(sid, \ell, \Gamma_i, \{G_j, D_j, \psi_{j,i}, \hat{\psi}_{j,i}\}_j, \delta_i, k, \chi)$.

Errors. Upon receiving a complaint from any other $\mathcal{P}_j \in \boldsymbol{P}$, report a complaint and halt.

Figure 25: ECDSA Point-Reveal

F.1 Missing ZK-Proofs for Pre-Signing w/o Point-Reveal

Paillier Decryption modulo q. For Pallier public key N_0 and prime q, the following relation verifies that the plaitext-value of Paillier ciphertext C is equal to x modulo q. Define

$$R_{\mathsf{dec}} = \left\{ (N_0, q, C, x; y, \rho) \mid x = y \mod q \ \wedge \ C = (1 + N_0)^y \cdot \rho_0^N \in \mathbb{Z}_{N_0^2}^* \right\}.$$

Paillier Multiplication. For Pallier public key N_0, the following relation verifies that the plaitext-value of Paillier ciphertext D is equal to the multiplication of the plaintext-values of C_1 and C_2 lying in range $\mathcal{I}$. Define

$$R_{\mathsf{mul}} = \left\{ (N_0, D, C_1, C_2, \mathcal{I}; x_1, \rho_1, x_2, \rho_2, \mu) \mid x_1, x_2 \in \mathcal{I} \ \wedge \ \{C_i = (1 + N_0)^{x_i} \cdot \rho_i^{N_0}\}_{i=1,2} \ \wedge \ D = (1 + N_0)^{x_1 x_2} \mu^{N_0} \right\}.$$

FIGURE 26 (Paillier Multiplication)

- **Inputs:** Common input is (N, X, Y, C).
  The Prover has secret input $(x, y, \rho, \rho_x, \rho_y)$ such that $x \in \pm 2^{\ell}$, $y \in \pm 2^{\ell'}$, $(1+N)^x \rho_x^N = X$ and $(1+N)^y \rho_y^N = Y$ and $C = (1+N)^{xy} \cdot \rho^N \mod N^2$.

1. Prover samples
$$\begin{cases} \alpha, r \leftarrow \mathbb{Z}_N^*, \\ s \leftarrow \mathbb{Z}_N^* \end{cases} \quad \text{and computes} \quad \begin{cases} A = Y^{\alpha} \cdot r^N \mod N^2 \\ B = (1+N)^{\alpha} s^N \mod N^2 \end{cases}$$
   and sends (A, B) to the verifier.
2. Verifier replies with $e \leftarrow \pm q$.
3. Prover Prover sends (z, u, v) to the verifier where $z = \alpha + ex$, $u = r \cdot \rho^{-z} \cdot \rho^e$, and $v = s \cdot \rho_x^e \mod N$.

- **Equality Checks**: $Y^z u^N = A \cdot C^e \mod N^2$ and $(1+N)^z c^N = B \cdot X^e \mod N^2$

Figure 26: Paillier Multiplication

FIGURE 27 (Paillier Decryption modulo q)

- **Setup:** Auxiliary RSA modulus $\hat{N}$ and Ring-Pedersen parameters $(s, t) \in \mathbb{Z}_{\hat{N}}^{*2}$.
- **Inputs:** Common input is (q, N_0, C, x).
  The Prover has secret input y, ρ such that $C = (1+N_0)^y \cdot \rho^{N_0} \mod N_0^2$ and $x = y \mod q$.

1. Prover samples $\alpha \leftarrow \pm 2^{\ell+\varepsilon}$ and
$$\begin{cases} \mu \leftarrow \pm 2^{\ell} \cdot \hat{N},\ \nu \leftarrow \pm 2^{\ell+\varepsilon} \cdot \hat{N} \\ r \leftarrow \mathbb{Z}_N^* \end{cases} \quad \text{and computes} \quad \begin{cases} S = s^y t^{\mu},\ T = s^{\alpha} t^{\nu} \mod \hat{N} \\ A = (1+N_0)^{\alpha} r^{N_0} \mod N_0^2 \\ \gamma = \alpha \mod q \end{cases}$$
   and sends (A, γ) to the verifier.
2. Verifier replies with $e \leftarrow \pm q$
3. Prover sends (z_1, z_2) where
$$\begin{cases} z_1 & = \alpha + ey \\ z_2 & = \nu + e\mu \\ w & = r \cdot \rho^e \mod N_0 \end{cases}$$

- **Equality Checks**:
$$\begin{cases} (1+N_0)^{z_1} \cdot w^{N_0} = A \cdot C^e \mod N_0^2 \\ z_1 = \gamma + ex \mod q \\ s^{z_1} t^{z_2} = T \cdot S^e \mod \hat{N} \end{cases}$$

Figure 27: Paillier Decryption modulo q