UC Non-Interactive, Proactive, Distributed ECDSA with Identifiable Aborts*

Ran Canetti[†‡] Rosario Gennaro[§] Steven Goldfeder[¶]
Nikolaos Makriyannis[‖] Udi Peled[**]

October 21, 2024

Abstract

We present a distributed ECDSA protocol, for any number of signatories. The protocol improves on that of the authors (CCS'20), which in turn builds on the Gennaro & Goldfeder and Lindell & Nof protocols (CCS '18). Specifically:

- Only the last round of the protocol requires knowledge of the message, and the other rounds can take place in a preprocessing stage, lending to a *non-interactive* threshold ECDSA protocol.
- The protocol withstands adaptive corruption of signatories. Furthermore, it includes a periodic refresh mechanism and guarantees proactive security.
- The protocol achieves accountability by identifying corrupted signatories in case of failure to generate a valid signature. (Identifiable abort)

Furthermore, we formulate a distributed signature ideal functionality within the UC framework that guarantees unforgeability, proactive security, and identifiable abort, and show that the protocol realizes this functionality in the global random oracle model, assuming Strong RSA, DDH, semantic security of the Paillier encryption, and a somewhat enhanced variant of existential unforgeability of ECDSA.

This combination of properties (low latency, compatibility with cold-wallet architectures, proactive security, identifiable abort and universally composable security) make our protocol a good fit for threshold wallets for ECDSA-based cryptocurrencies.

*An extended abstract of this work appears in the proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS'20) [21]. This is an updated version.

[†]Authors are listed in alphabetical order.
[‡]Boston University. Member of CPIIS. canetti@bu.edu
[§]The City College of New York. rosario@ccny.cuny.edu
[¶]Cornell Tech/Offchain Labs. goldfeder@cornell.edu
[‖]Fireblocks. nikos@fireblocks.com
[**]Work done while at Fireblocks. udi0peled@gmail.com

Contents

1 Introduction

Introduced by Desmedt [31] and Desmedt and Frankel [32], distributed 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 all of the signatories agree to sign it. More specifically, a set of signatories, presented with a message m, jointly and interactively compute a *signature* σ such that (1) if all signatories agree to sign m, then the pair m, σ is accepted as valid by a pre-determined public verification algorithm, and (2) no attacker that controls all but one of the 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. (A natural generalization of distributed signatures involves enabling only privileged subsets of signatories to sign. However, we focus on the more basic and prevalent case of plain distributed signature schemes, which, by synecdoche, we refer to as 'threshold signature' schemes, as is commonly done in the cryptography literature.)

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 that involves using a secret key, by distributing the capabilities that require using the secret key among multiple servers/devices, and guaranteeing security as long as at least one server/device remains uncorrupted. Indeed, this approach eliminates the single point of failure when using the secret key. Examples include threshold El-Gamal, RSA, Schnorr, Cramer-Shoup, and others [29, 58, 60, 59, 17].

With the advent of blockchain technologies and cryptocurrencies over the past decade, there has been a strong renewed interest in threshold cryptography, particularly in threshold signatures. As digital signatures are essential for enabling transactions, many stakeholders are seeking ways to perform signature generation in a distributed manner. Consequently, numerous companies are now offering solutions based on, or in combination with, threshold cryptography.[1]

Threshold ECDSA. The *digital signature algorithm* (DSA) [49] in its elliptic curve variant (ECDSA) [56] 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. Early attempts towards making threshold DSA/ECDSA practically efficient include Gennaro et al. [40] in the honest majority setting and MacKenzie and Reiter [53] in the two-party setting.

In recent years, numerous protocols for threshold ECDSA [41, 7, 38, 50, 52, 33, 34, 28, 23, 24, 21, 45, 35, 25] have been developed, supporting any number n of parties and allowing any threshold $t - 1 < n$ of corrupted parties. These recent protocols generally achieve competitive practical performance, with varying trade-offs between computation and communication costs depending on the tools used. It is worth noting that all protocols prior to [35] required at least four communication rounds (or 8 prior to [21]), which can be a significant bottleneck in many modern communication settings, particularly when involving geographically dispersed servers or mobile devices. A comparison of the complexity between the aforementioned protocols and our protocol is provided in the following section.

Another limitation of prior work relates to the type of security analysis provided. Specifically, security was proved either via an ad-hoc game-based extenstion of the traditional notion of existantial unforgeability (as in, e.g. [38, 23, 24]), or by way of showing that the protocol securely evaluates the ECDSA function within the universally compoasble (UC) security framework, e.g. as in [51, 50, 52, 33, 34, 35]. While the former modeling does not provide rigorous composable security guarantees for applications using the protocol, the latter modeling turns out to be hard to work with; for instance, works that take this approach are unable to guarantee security when party corruptions happen over time based on the execution history.

1.1 Our Results.

We present a new threshold ECDSA protocol. The protocol builds on the authors' previous work [21], which, in turn, builds on the techniques from Gennaro and Goldfeder [38] and Lindell et al. [52]. Our protocol offers improved efficiency and security guarantees compared to Gennaro and Goldfeder [38], Lindell et al. [52], Canetti et al. [21], and remains competitive with other state-of-the-art ECDSA protocols. We first discuss the new

[1] See https://www.mpcalliance.org/ for companies in the threshold cryptography space (retrieved August 2024).

characteristics of the protocol at a high level, followed by a detailed description of its construction and analysis. Figure 1 provides a rough comparison of the main costs and security guarantees between our protocol and those mentioned in the previous section. See Section 9 for additional discussion of related work.

Signing Protocol	*Rounds*	*Group Ops*	*Ring Ops*	*Communication*	*Proactive*	*ID Abort*	*UC*$^{\heartsuit}$
Gennaro and Goldfeder [38]	9	$10n$	$50n$	$10\kappa + 20\nu$ (7 KiB)	✗	✗	✗
Lindell et al. [52] (Paillier)†‡	8	$80n$	$50n$	$50\kappa + 20\nu$ (7.5 KiB)	✗	✗	✓
Lindell et al. [52] (OT)†	8	$80n$	0	$20\kappa^2$ (190 KiB)	✗	✗	✓
Doerner et al. [34]	$\log(n)$	5	0	$10\kappa^2$ (90 KiB)	✗	✗	✓
Castagnos et al. [24]*	8	$15n$	0	100κ (3.5 KiB)	✗	✗	✗
Canetti et al. [21]	4 or 7	$10n$	$90n$	$10\kappa + 50\nu$ (15 KiB)	✓	✓	✓
Castagnos et al. [25]*	4	$15n$	0	100κ (3.5 KiB)	✓	✓	✗
Haitner et al. [45]	5	$50n$	0	$8\kappa^2$ (60 KiB)	✗	✗	✓
Doerner et al. [35]	3	$6n$	0	$6\kappa^2$ (50 KiB)	✗	✗	✓
This Work: *Interactive*§	3	$25n$	$75n$	$65\kappa + 50\nu$ (15 KiB)	✓	✓	✓
This Work: *Non-Interactive*§	$3 + 1$	$25n$	$75n$	$65\kappa + 50\nu$ (15 KiB)	✓	✓	✓

Figure 1: Approximate comparison of our scheme with related works for the signature generation phase. Costs are displayed per party for a n-party protocol secure against $n - 1$ corrupted parties as reported in each corresponding work, or inferred from the protocol descriptions when not directly available. Ring operations contain two types of operations (mod N and N^2) that we do not distinguish in the table; operations modulo N^2 represent no more than a third of the total number of ring operations for all protocols. The communication column describes the number of group elements (encoded by κ bits) and ring elements (encoded by $\nu = \log(N)$ bits) sent between each pair of parties; in parentheses, we provide concrete estimates with $\kappa = 256$ for the curve secp256k1 (i.e., the Bitcoin curve [55]) and $\log(N) = \nu = 2048$ for 112-bit security for Paillier. (†Reported costs are based on Lindell et al. [52, Version 2018] which is subsumed by the updated paper [45]. ‡There is a discrepancy between the ring operations in the table above and [52] because of how operations mod N or N^2 are accounted for. *We note that [24] and [25] rely on somewhat incomparable hardness assumptions, and it involves operations in a different group than that of the underlying elliptic curve—cf. Section 9. §We report two round-complexities for our work to reflect the two different variants of our protocol. $^{\heartsuit}$The UC column indicates whether the corresponding work provides a security analysis in the UC framework. Yet, given the results presented in this paper, all of the protocols mentioned above UC-realize the generic threshold signature functionality described here.) All works are listed in chronological order.

Non-interactive signing. The ability to generate signatures non-interactively—where each signatory can independently produce a signature share for a given message without interacting with other signatories, and a public algorithm can then combine these shares into a single signature—is an extremely useful feature. This capability is particularly valuable in scenarios such as using a 'cold wallet' mechanism for some of the signatories, a common practice in the digital asset space. Notably, several popular signature schemes, such as RSA [47] and BLS [6], support threshold protocols with non-interactive signing.

While no completely non-interactive threshold ECDSA signature generation is known, it was observed by Dalskov et al. [28] that the structure of the ECDSA signature allows for doing much of the work ahead of time, before the message to be signed is known. In [21], this idea is further developed by splitting the signing process into two phases: an initial preprocessing phase that requires three rounds and can be completed before the message is known, followed by a non-interactive step where each signatory generates their own signature share once the message to be signed is known. This second step involves the computation and transmission of only a single field element (256 bits for the Bitcoin curve). The protocol presented here retains this feature from [21] while significantly simplifying the overall protocol.

Round-minimal interactive signing. For the fully online variant (i.e. the interactive case), our protocol matches the round complexity of Doerner et al. [35] (three rounds) by appropriately modifying the third-round messages in the pre-processing phase to depend on the message being signed, so that they yield suitable signature shares. This is explained in Section 2.5.

Proactive key refresh [57, 46, 18, 48]. While threshold signatures do provide a significant security improvement over plain signature schemes, they may still be vulnerable to attacks that, over time, compromise all shareholders. 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 the signatories periodically engage in a key-refresh protocol. We then call the period of time starting at the onset of one key refresh, and ends at the conclusion of the following key refresh, an *epoch*. The security guarantee is that the scheme remains unforgeable as long as for each epoch there is at least one signatory that is uncorrupted *throughout that epoch*.

It is stressed that the public signature verification key of the scheme remains the same throughout. At the same time, the ability to generate signatures is only preserved when all signatories (even compromised ones) follow their protocols and their local data is preserved. Any single corruption of the state of a server, as well as any divergence from the protocol, will result in losing its ability to generate signatures. (While there are standard ways to 'harden' the key refresh protocol so as to guarantee continued generation of signatures even when some of the signatories are actively corrupted [57], we keep such extensions out of scope for this work.) Our protocol offers a three-round key refresh phase.

Identifiable aborts. Our protocol contains an additional mechanism that allows the signatories to identify, should the parties fail to obtain a valid signature on a given message, at least one out of the signatories that has failed to participate in the generation of the signature, or has otherwise deviated from the prescribed protocol. In this paper, we refer to this property, commonly known as 'identifiable aborts' in the MPC literature, as *accountability*.

The communication model. We assume that the signatories are connected via an authenticated and synchronous broadcast mechanism. That is, the communication is public, and every message that is sent will be received by all parties in the next round. While the protocol can be potentially hardened to deal with other communication models, this basic model appears to both capture the essence of the problem and reflect the common assurances in threshold signature applications.

Security & Composability. We provide a security analysis of our protocol within the Universally Composable (UC) Security framework [16]. However, unlike prior work we do not insist on demonstrating that our protocol securely evaluates any specific function. Instead, we formulate an ideal threshold signature functionality that provides an ideal signing service that offers unconditional unforgeability, periodic key refresh, and identifiable abort, against an adversary that adaptively corrupts signatories, as long as at least one signatory remains uncorrupted between consecutive refreshes. We then show (under hardness assumption detailed below) that our protocol realizes this functionality, while guaranteeing that legitimate signatures are verifiable by the standard ECDSA verification algorithm. This modeling and analysis allows benefitting from the composability guarantees provided by the UC framework (which is of particular importance in applications such as cryptocurrencies, where the distributed signature module is a component of a larger system that is expected to provide strict assurance guarantees), while not being overly restrictive.

Our analysis uses the strict variant of the global random oracle model (GROM) of [20, 13]. (Incorporating the random oracle as a global functionality within the plain UC framework is done using [2].)

Cryptographic assumptions and 'enhanced unforgeability'. We show that our ptotocol securely realizes the above ideal theshold signature functionality under the following assumptions. We assume the unforgeability of plain, non-threshold ECDSA, the semantic security of Paillier encryption (DCR), Decisional Diffie–Hellman (DDH), and strong-RSA. For the non-interactive variant, we require a somewhat enhanced unforgeability property of ECDSA, which accounts for scenarios where the adversary obtains some 'leakage' information on the random string that the signer will use for generating the upcoming signatures. Still, the adversary should not be able to forge signatures, even given this leakage. We call this property *enhanced*

unforgeability, and demonstrate that ECDSA is enhanced unforgeable in the generic group model when the ECDSA digest function is modelled as a random oracle.

1.2 Paper Organization

The paper is organized as follows. In Section 2, we provide a high-level yet detailed overview of our techniques, including protocol descriptions, the NIZK technique, and the security analysis. This analysis is further broken down into the description of the ideal functionality and the demonstration that our protocol realizes said ideal functionality. .

Section 3 covers the necessary toolbox, including the cryptographic primitives and the communication model that will be used in subsequent sections. Sections 4 and 5 contain, respectively, the protocol description and the descriptions of the underlying zero-knowledge (ZK) protocols that give rise to the non-interactive zero-knowledge proofs (NIZKs) used extensively in the protocol. Section 6 contains the description of the random oracle model, the ideal functionality, and our main security claims. Section 7 is dedicated to the proof of Theorem 6.6, which is the cornerstone of our security analysis. Finally, Section 8 describes how to handle adaptive corruptions and provides a security reduction from the adaptive to the static adversarial model for our protocol.

Section 9 provides additional discussion on related works, including those that appeared after the authors' CCS 2020 publication [21].

Appendix organization. The appendix is organized as follows. Appendix A contains ZK protocols that are very similar to the ones presented in Section 5 or are standard (e.g., Schnorr proof of knowledge of discrete logarithm). Appendix B contains the proof of Theorem 6.7, which is similar to the proof of Theorem 6.6 in Section 7. Appendix C provides our theoretical complexity estimates as well as a discussion on concrete security. Appendix D contains straightforward facts from elementary number theory and probability theory. Appendix E presents our security proof of enhanced unforgeability in the GGM. The paper concludes with Appendix F, discussing the extension to the t-out-of-n threshold case and providing comments on the previous version of the key-refresh protocol.

2 Overview of our Techniques

Hereafter, $\mathbb{F}_q$ denotes the finite field with q elements and $\mathcal{F} : \boldsymbol{M} \to \mathbb{F}_q$ 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.

Recall that an ECDSA signature for secret key $x \in \mathbb{F}_q$ and message msg has the form $(\rho, \gamma^{-1} \cdot (m + \rho x)) \in \mathbb{F}_q^2$, where $m = \mathcal{F}(\text{msg})$, ρ is the x-axis projection of the point $g^\gamma \in \mathbb{G}$, and γ is a uniformly random element of $\mathbb{F}_q$. The verification algorithm accepts a signature (ρ, σ) as valid for message msg $\in \boldsymbol{M}$ with respect to public key $X = g^x \in \mathbb{G}$, if ρ is the x-axis projection of $g^{m\sigma^{-1}} \cdot X^{\rho\sigma^{-1}}$, where $m = \mathcal{F}(\text{msg})$. As a matter of terminology, γ and $\Gamma = g^\gamma$ are referred to as the secret and public *nonce*, respectively.

2.1 Background [38]

As preliminary background, we start by describing the basic protocol from [38] in the honest-but-curious case, i.e. the case where all signatories follow the protocol specifications.

Key-generation phase. 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 $X = X_1 \cdot \ldots \cdot X_n \in \mathbb{G}$. 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 algorithms associated with $\mathcal{P}_i$. It is stressed that all parties can run the encryption algorithm.

Signing phase. To sign a message msg, the parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ generate local shares $\gamma_1, \ldots, \gamma_n$, respectively, of the random value $\gamma = \gamma_1 + \ldots + \gamma_n$, as well as local shares $k_1, \ldots, k_n$, respectively, of an ephemeral value $k = k_1 + \ldots + k_n$ which will be used to mask γ. 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 $\hat{\alpha}_{i,j} + \hat{\beta}_{j,i} = x_j k_i$. Namely, all pairs of parties engage in a share-computation phase such that each $\mathcal{P}_i$ interacting with $\mathcal{P}_j$ provides input $K_i = \mathsf{enc}_i(k_i)$ and obtains output $(D_{i,j}, \hat{D}_{i,j}) = (\mathsf{enc}_i(\gamma_j k_i - \beta_{j,i}), \mathsf{enc}_i(x_j k_i - \hat{\beta}_{j,i}))$.[2]

Next, each $\mathcal{P}_i$ sets $\alpha_{i,j} = \mathsf{dec}_i(D_{i,j})$ and $\hat{\alpha}_{i,j} = \mathsf{dec}_i(\hat{D}_{i,j})$, and the share-computation phase terminates by aggregating the outputs across all parties; each $\mathcal{P}_i$ calculates $\delta_i = \gamma_i k_i + \sum_{j \neq i} \alpha_{i,j} + \beta_{i,j}$ and $\chi_i = k_i x_i + \sum_{j \neq i} \hat{\alpha}_{i,j} + \hat{\beta}_{i,j}$.[3] Along the way, i.e. concurrently with the share-computation phase, each $\mathcal{P}_i$ sends $\Gamma_i = g^{\gamma_i}$ and the parties set $\rho = \left(\prod_j \Gamma_i\right)|_{\text{x-axis}}$. Finally, each $\mathcal{P}_i$ sends $(\delta_i, \hat{\sigma}_i)$ to all, where $\hat{\sigma}_i = k_i \cdot m + \rho\chi_i$, where $m = \mathcal{F}(\text{msg})$ is the hash-value of msg. The signature is set to (ρ, σ) for $\sigma = (\sum_j \hat{\sigma}_j) \cdot (\sum_j \delta_j)^{-1} = \gamma^{-1}(m + \rho x)$.

Summary. The basic protocol proceeds as follows from $\mathcal{P}_i$'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 $(D_{j,i}, \hat{D}_{j,i}, \Gamma_i = g^{\gamma_i})$ to each $\mathcal{P}_j$.
3. When obtaining $\{(D_{i,j}, \hat{D}_{i,j}, \Gamma_j)\}_{j \neq i}$, set δ_i, σ_i as prescribed, and send (δ_i, σ_i) to all.

Output. When obtaining $\{\delta_i, \hat{\sigma}_j\}_{j \neq i}$, set ρ as prescribed, and output (ρ, σ) for $\sigma = (\sum_j \hat{\sigma}_j) \cdot (\sum_j \delta_j)^{-1}$.

The above protocol takes three 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 information is exposed about x as the transcript of the protocol until-and-excluding round three is independent of the secret key x. Finally, revealing $(\delta_i, \hat{\sigma}_i)$ is equivalent to revealing the signature (ρ, σ), i.e. no information is leaked beyond the signature itself.

However, if a corrupted party deviates from the specification of the protocol, it may cause other parties to send third-round messages that leak information about the secret key; potentially the entirety of it. To mitigate this problem, Gennaro and Goldfeder [38] devise a special-purpose, clever technique that allows the parties to verify the validity of the signature shares before releasing them. However, their protocol ends up adding six rounds of communication.

2.2 Our Approach

Using the above basic protocol as a starting point, we show how the parties can verify the validity of the signature shares without adding any rounds, and at a comparable computational cost to that of [38]. 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 third and last round. This, in turn, leads to our non-interactive variant. To achieve *cheap* accountability in the non-interactive variant of our protocol, we require one additional round during the pre-processing phase, bringing the total round count to four, instead of three. Proactive key refreshes are also naturally built on top of the basic protocol, with appropriate non-interactive zero-knowledge proofs (henceforth referred to as NIZKs).

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. We refer the reader to the subsequent technical sections for further

[2] 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 is an encryption of k_i under their own public key. Then, $\mathcal{P}_j$ samples a random $\beta_{j,i}$ from a suitable range, and, using the homomorphic properties of the encryption scheme, $\mathcal{P}_j$ computes $D_{i,j} = (\gamma_j \odot K_i) \oplus \mathsf{enc}_i(-\beta_{j,i})$, where $\oplus$ and $\odot$ denote homomorphic evaluation of addition and (scalar) multiplication (with proper rerandomization), respectively. So, $D_{i,j}$ is an encryption of $\gamma_j k_i - \beta_{j,i}$ under $\mathcal{P}_i$'s public key.

[3] Notice that $\delta_1, \ldots, \delta_n$ and $\chi_1, \ldots, \chi_n$ are additive sharings of γk and xk, respectively.

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. In this high-level overview, when we say that a party broadcasts a message, we mean that this message is received by all other parties. We will refine the meaning of 'broadcast' in the subsequent technical sections.

Key-generation. As in the basic protocol, each $\mathcal{P}_i$ samples a local random secret share $x_i \leftarrow \mathbb{F}_q$ of the (master) secret key $x = \sum_i x_i$, it produces a group element $X_i = g_i^x$, and then broadcasts a commitment to X_i. Once receiving the commitments from every other party, it broadcast-decommits X_i and also a Schnorr NIZK of x_i (the Discrete Log 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 along with a NIZK proof that N_i is well-constructed (i.e., that it is the product of suitable primes). Next, each $\mathcal{P}_i$ selects a random secret sharing of $0 = \sum_j x_{i,j}$ and computes $X_{i,j} = g^{x_{i,j}}$ for every $j \neq i$. Then, $\mathcal{P}_i$ broadcasts $(X_{i,j})_j$ along with a NIZK proof of the discrete logarithms of $\{X_{i,j}\}_j$, and sends the discrete logarithm $x_{i,j}$ of $X_{i,j}$ to $\mathcal{P}_j$ for every $j \neq i$. The parties update their key shares by setting $x_i^* = x_i + \sum_k x_{k,i} \mod q$, provided that all the proofs are valid and $\prod_k X_{j,k} = \mathsf{id}_{\mathbb{G}}$ for every j.

Pre-processing (aka Presigning). Ahead of time, before the message to be signed is known, the parties can choose to generate 'presignature' shares through the following presigning process. Each party is instructed to commit to γ_i and k_i by encrypting these values under their own keys and broadcasting $G_i = \mathsf{enc}_i(\gamma_i)$ and $K_i = \mathsf{enc}_i(k_i)$. Concurrently, the parties initiate a pairwise share-computation phase (for $x_j k_i = \alpha_{i,j} + \beta_{j,i}$ and $\gamma_j k_i = \hat{\alpha}_{i,j} + \hat{\beta}_{j,i}$) and a public nonce computation phase (for $\Gamma = g^\gamma \in \mathbb{G}$, corresponding to the nonce of the future signature). During these phases, the parties also prove in zero-knowledge that the values used in the multiplications are consistent with the values encrypted in G_i, K_i, as well as the exponent of the public key-share $X_i = g^{x_i}$.

Finally, when the aforementioned processes terminates, the parties spend an additional round to compute the field element $\delta = \gamma \cdot k \in \mathbb{F}_q$, which corresponds to the value of the secret nonce γ masked by a random field element $k = \sum_j k_i$. This step includes suitable zero-knowledge proofs to ensure that the parties obtain the correct value of δ. At the end of the presigning phase, each $\mathcal{P}_i$ stores in memory the tuple $(k_i/\delta, \chi_i/\delta, \Gamma)$, where k_i/δ represents the share of γ^{-1} (since $\delta = k \cdot \gamma$ and $\sum_i k_i = k$), χ_i/δ represents the share of $\gamma^{-1} \cdot x$ (since $\delta = k \cdot \gamma$ and $\sum_i \chi_i = k \cdot x$), and $\Gamma = g^\gamma \in \mathbb{G}$ is the public nonce.

Remark 2.1. A key innovation in our work is using the Paillier cryptosystem as a commitment scheme, setting us apart from other Paillier-based protocols like [38]. Encrypting values under the parties' own public keys provides a binding and hiding commitment scheme, assuming Paillier's semantic security. This approach enables straightline extraction of the adversary's secrets in the security analysis, as the corrupted parties' Paillier keys are obtained in the auxiliary information phase.

In summary, by virtue of the 'Paillier commitments' and the accompanying NIZKs, party $\mathcal{P}_i$ is confident that the tuple $(\Gamma, k_i/\delta, \chi_i/\delta)$ is well-formed at the end of presigning phase, and there is no need for additional communication rounds to verify the correctness of the tuple, as opposed to [38, 52, 34, 24].

Non-Interactive signing. Once a message msg is known, to generate a signature for presigning data $(\Gamma, k_i/\delta, \chi_i/\delta)$, each $\mathcal{P}_i$ sets $m = \mathcal{F}(\text{msg})$, computes $\rho = \Gamma|_{(\cdot)}$, and sends $\sigma_i = k_i m/\delta + \rho\chi_i/\delta \mod q$ to all parties. After receiving the other parties' signature-shares, the parties output the signature $(\rho, \sum_i \sigma_i) = (\rho, \gamma^{-1}(m + \rho x))$. We note that that the parties detect cheating at this step if the resulting string fails to verify.

Managing Presignature Tuples For a given presigning parameter $L \in \mathbb{N}$, after the parties run the presigning phase L times concurrently, they obtain presigning data $\{(\ell, \Gamma_\ell, \tilde{k}_i^\ell, \tilde{\chi}_i^\ell)\}_{\ell=1,\ldots,L}$. Proper management of this data is crucial, as any mishandling could potentially lead to key leakage. First, the presigning tuple $(\ell, \Gamma_\ell, \tilde{k}_i^\ell, \tilde{\chi}_i^\ell)$ must be erased immediately after it is used. Second, during key refresh, any unused presignatures must be discarded, i.e. erased.[4]

[4] Alternatively, it is possible to retain the presigning data between key refreshes as long as it is properly refreshed, i.e. by re-randomizing $(\tilde{k}_i, \tilde{\chi}_i)$. However, we emphasize that any presigning tuple, whether re-randomized or not, must be erased

2.4 Identifying Corrupted Parties

Whenever a NIZK fails for a party, the remaining parties attribute the fault to that party and report it as corrupted. The challenge arises in identifying corrupted parties when δ is malformed or the signature string does not verify, and yet all the NIZKs verify. To address these situations, we introduce a special-purpose process, which is outlined below.

Accountability during signing. We instruct the parties to publish $(\tilde{\Delta}_i, \tilde{S}_i) = (\Gamma^{k_i/\delta}, \Gamma^{\chi_i/\delta})$ during the presigning phase, and $\mathcal{P}_i$ saves $\{(\tilde{\Delta}_j, \tilde{S}_j)\}_{j\neq i}$ for the signing phase later on. During the signing phase, the signature share $\sigma_i = (k_i/\delta) \cdot m + \rho \cdot (\chi_i/\delta)$ can be verified in the exponent using the formula $\Gamma^{\sigma_i} = \tilde{\Delta}_i^m \cdot \tilde{S}_i^\rho$.

Therefore, assuming that the set $\{(\tilde{\Delta}_j, \tilde{S}_j)\}_{j=1}^n$ is well-formed, the correctness of each σ_i can be verified using the tuple $(\Gamma, \tilde{\Delta}_i, \tilde{S}_i)$. The remaining task is to ensure that the set $\{(\tilde{\Delta}_j, \tilde{S}_j)\}_{j=1}^n$ is indeed well-formed, as explained below.

Accountability during presigning. Each $\mathcal{P}_i$ publishes $(\Delta_i, \tilde{S}_i) = (\Gamma^{k_i}, \Gamma^{\chi_i})$ in the last round of the presigning phase, along with a NIZK that $\Delta_i = \Gamma^{k_i}$. Notice that if $\prod_{j=1}^n \Delta_i = g^\delta$, then δ is well formed as $\prod_{j=1}^n \Delta_i = \Gamma^{\sum_{j=1}^n k_i}$ which implies that $\delta = k \cdot \gamma$. If the check $\prod_{j=1}^n \Delta_j = g^\delta$ fails, each $\mathcal{P}_i$ is instructed to prove that they sent the correct δ_i by opening the relevant ciphertexts and verifying algebraically whether the correct data was sent.[5] Specifically, if δ is malformed, the parties open the ciphertexts resulting from the pairwise multiplication of the k's and the γ's (i.e. $K_i, G_i, D_{i,j}$). At this point, all honest parties can agree on who sent the incorrect value.

Similarly, the parties can verify that the S_i's are well-formed by checking $\prod_j S_j = X^\delta$. If this check fails, each party proves that their own S_i is well-formed, i.e. that it equals Γ^{χ_i}, where χ_i is the secret value obtained from the pairwise multiplication of the k's and the x's. This process is similar to checking that δ_i is well-formed, except that the value χ_i remains in the exponent and is not revealed in plaintext.

In the end, if no errors are detected, the parties store the set $\{(\tilde{\Delta}_j, \tilde{S}_j) = (\Delta_j^{\delta^{-1}}, S_j^{\delta^{-1}})\}_{j=1}^n$ for use in the signing phase.

2.5 Two-round Presigning

When accountability is not a concern, or it can be enforced with a storage penalty during signing, it is possible to recover the round-complexity of the basic three-round protocol described in Section 2.1. Specifically, the presigning stage of the protocol terminates after round two, and the parties are instructed to store the tuple $(\Gamma, k_i, \chi_i, \delta_i)$ instead of $(\Gamma, k_i/\delta, \chi_i/\delta)$. Consequently, the parties are not required to spend resources verifying the validity of $\delta = \sum_j \delta_j$.

During signing, once $m = \mathcal{F}(\text{msg})$ is known, the parties simply broadcast $(\delta_i, \hat{\sigma}_i = k_i m + \rho\chi_i)$ and the parties output (ρ, σ), where $\sigma = (\sum_i \hat{\sigma}_i) \cdot (\sum_i \delta_i)^{-1}$ if and only if (ρ, σ) is a valid signature for msg. In case of failure, the parties are instructed to prove the well-formedness of the pair $(\delta_i, \hat{\sigma}_i)$ with respect to the transcript of the presigning phase; this solution requires that the parties save the transcript of the presigning phase for the signing phase. The process is similar to the process described in Section 2.4, and a full description is omitted from the current paper. The parties ignore the last step if accountability is not desired.

2.6 Security Analysis

As mentioned above, we prove security of the protocol by first formulating an ideal functionality, $\mathcal{F}_{\mathsf{tsig}}$, for threshold signature schemes within the universally composable (UC) security model [16], and then demonstrating that our protocol securely realizes $\mathcal{F}_{\mathsf{tsig}}$. This approach allows us to keep the protocol relatively lean and simple, and at the same time provide strong and nuanced composable security guarantees against powerful adversaries.

---

immediately after it is consumed.

[5] For security reasons, the opening of the ciphertext is done in zero-knowledge by revealing g^μ—only the plaintext μ in the exponent is revealed, not the plaintext itself or the randomizer.

2.6.1 Ideal Functionality $\mathcal{F}_{\mathsf{tsig}}$

We give a high-level description of the operation of $\mathcal{F}_{\mathsf{tsig}}$. At invocation, $\mathcal{F}_{\mathsf{tsig}}$ is provided with a distinguished set of parties (i.e. the signatories), which are initially marked as uncorrupted. Then:

1. When the adversary takes control of a party, $\mathcal{F}_{\mathsf{tsig}}$ marks the party as corrupted.
2. When a key refresh occurs, any parties that were previously marked as corrupted but are no longer under the adversary's control are marked as uncorrupted again.
3. When all parties request the generation of a formal public verification algorithm, $\mathcal{F}_{\mathsf{tsig}}$ obtains the algorithm $\mathcal{V}$ from the adversary and returns it to the parties.
4. When all parties request the generation of a signature string for a message m, $\mathcal{F}_{\mathsf{tsig}}$ obtains a signature string σ from the adversary and returns it. If the provided verification algorithm fails to verify the signature for this message (i.e. if $\mathcal{V}(m, \sigma) = 0$), $\mathcal{F}_{\mathsf{tsig}}$ discloses the identity of a corrupted party. This provision ensures accountability.
5. When anyone (not necessarily one of the signatories) requests verification of a signature σ with respect to $\mathcal{V}$ and a message m, $\mathcal{F}_{\mathsf{tsig}}$ proceeds as follows: If there is a currently uncorrupted party that has not requested to sign m, $\mathcal{F}_{\mathsf{tsig}}$ responds with a verification failure. Otherwise, $\mathcal{F}_{\mathsf{tsig}}$ responds according to $\mathcal{V}(m, \sigma)$. This provision ensures proactive unforgeability against mobile, adaptive corruptions.

Indeed, $\mathcal{F}_{\mathsf{tsig}}$ can be thought of as an idealized signature service that applications can rely on, providing guarantees of (i) unforgeability against adaptive corruptions, as long as at least one party remains uncorrupted between any two consecutive key refreshes, and (ii) accountability in cases where the signature generation process fails to produce a valid signature. The universal composition theorem [16] ensures that these same security properties are preserved when replacing $\mathcal{F}_{\mathsf{tsig}}$ with any protocol that UC-realizes it.

Unforgeability and accountability as trace properties. It is important to note that protocols that UC-realize $\mathcal{F}_{\mathsf{tsig}}$ are not necessarily designed to securely evaluate or sample the ECDSA function, or any other specific function. Instead, the guarantees provided by $\mathcal{F}_{\mathsf{tsig}}$ are best understood as 'trace properties'—meaning that any violation of these guarantees can be easily detected by examining the trace of the execution prefix that led to the violation.[6] This is a significant relaxation relative to 'secure evaluation' properties that typically include also requirements related to hiding of input or output values of the computation. Our security analysis exploits this feature of $\mathcal{F}_{\mathsf{tsig}}$ as follows.

Recall that a security proof showing that a protocol π UC-realizes an ideal functionality $\mathcal{F}$ involves two key steps. First, an ideal model adversary (a 'UC simulator') $\mathcal{S}$ is described. This simulator interacts with $\mathcal{F}$ on one side and with the UC environment $\mathcal{Z}$ on the other, simulating an interaction with π. The next step is to argue that this simulation is valid, meaning that $\mathcal{Z}$ cannot distinguish between the simulated execution and an actual interaction with π.

2.6.2 UC Simulator

We use a very simple UC simulator, which simply computes the messages sent by each uncorrupted party $\mathcal{P}$ according to the actual protocol instructions. Similarly, when a party is corrupted, $\mathcal{S}$ reports the actual internal state of the party to $\mathcal{Z}$.

This simple simulation strategy is possible since $\mathcal{F}_{\mathsf{tsig}}$ reveals the input values obtained from the parties to $\mathcal{S}$. Additionally, this simplistic simulation is *perfect*—when limited to executions where no violations of unforgeability or accountability occur.

[6] Specifically, a violation would be either (i) a verification request for signature σ on message m, such that not all parties asked to sign m and yet $\mathcal{V}(m, \sigma) = 1$, or (ii) a signing request for a message m that results in an invalid signature and yet there is no consensus on the corrupted party's identity among the honest parties.

Demonstrating the validity of the UC simulation. To establish the validity of the UC simulation, it remains to show that any environment machine $\mathcal{Z}$ can cause forgery or accountability-violation events to occur only with negligible probability. This is accomplished by reducing these events to the assumptions mentioned earlier (namely the unforgeability of plain, non-threshold ECDSA, the semantic security of Paillier encryption, DDH, and Strong RSA). Importantly, this reduction is not constrained by the limitations of the UC model; instead, it treats the environment machine $\mathcal{Z}$ as a plain family of polysize circuits. In particular, the reductions can rewind $\mathcal{Z}$ and also program the random oracle used by $\mathcal{Z}$.

Below we provide more details on these reductions. First, however, let us discuss a number of points regarding the overall approach.

Discussion on the UC Simulator. We emphasize that employing a 'trivial' UC simulator, which simply mirrors the protocol execution and then bounds the probability of bad events through reductions outside the UC model, is crucial for proving the security of our protocol. Indeed, in this context, the UC simulator has little choice but to perform this trivial simulation. For one, under the UC model, the simulator is restricted to straight-line simulation. Additionally, due to our use of the strict GROM, the simulator is unable to observe or program the oracle queries made by the environment.

At the same time, the fact that the reductions occur outside the formal UC model does not diminish the security guarantees provided by $\mathcal{F}_{\mathsf{tsig}}$, and the applicability of this 'trivial' simulation technique seems to extend beyond our protocol or even $\mathcal{F}_{\mathsf{tsig}}$. Specifically, consider an ideal functionality that is a *trace-property ideal functionality*, where the only security guarantee it provides is the absence of bad events that are trace properties, as defined above. (Formally, a trace-property ideal functionality immediately discloses all its inputs to the adversary and allows the adversary to determine all of its outputs, provided these outputs do not violate a set of trace properties.) In such cases, demonstrating that a protocol UC-realizes a trace-property ideal functionality can always use the same methodology.

Liveness. Finally, we note that $\mathcal{F}_{\mathsf{tsig}}$ does not guarantee any 'liveness' properties. Indeed, protocols that allow the adversary to arbitrarily delay the generation of public keys or signatures may well UC realize $\mathcal{F}_{\mathsf{tsig}}$, as long as they are uforgeable. Still it is straightforward to verify that our protocol always generates a public key and a signature string within a fixed number of communication rounds. Similarly, any execution of the key refresh protocol completes (either successfully or unsuccessfully) within a fixed number of rounds.

2.6.3 Unforgeability proof

We provide additional details regarding the reduction to the unforgeability of non-threshold ECDSA. Consider the following experiments involving a simulator attempting to simulate $\mathcal{Z}$'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 $\mathcal{Z}$).
   (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' NIZKs, 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 $\mathcal{Z}$ decides to corrupt the special party, then the experiment is reset to the last preceding key-refresh by rewinding $\mathcal{Z}$).

(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 instead.

(c) Every time an honest party is instructed to generate a Diffie-Hellman (DH) tuple, the simulator computes a random tuple 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 if $\mathcal{Z}$ successfully forges signatures in our protocol, then $\mathcal{Z}$ also succeeds in forging signatures in all three experiments described above. From the third experiment, we conclude that $\mathcal{Z}$ forges signatures for the plain (non-threshold) ECDSA signature scheme, contradicting its presumed security.

The first two experiments serve as stepping stones towards proving that $\mathcal{Z}$ forges in the third experiment. Specifically, the real execution and the first experiment are statistically close, assuming all NIZKs are sound and the simulator extracts the correct values. The first and second experiments are computationally close under the DDH assumption, provided the Paillier cryptosystem is semantically secure (DCR assumption). Finally, the second and third experiments are identical in a perfect sense.

Dealing with adaptive party corruptions. To demonstrate the validity of the reduction against adaptive party corruptions, it suffices to argue that Experiments 2 and 3 terminate. Under the DDH assumption, our analysis shows that both experiments terminate in time linear in the number of parties. Consequently, $\mathcal{Z}$ forges signatures in the third experiment, contradicting the presumed security of plain ECDSA.

Extension to two-stage signature generation. When signatures are computed via a two-step process—first generating a pre-signature, followed by a non-interactive signature generation once the message is known—we can no longer reduce a forgery event in the threshold case to a standard forgery event in plain ECDSA. Instead, we reduce a forgery event in the threshold case to an enhanced variant of ECDSA unforgeability. In this enhanced game, the adversary is provided access to a 'two-stage' signing oracle: first, the adversary requests the nonce $r = g^{\gamma}|_{(\cdot)}$, the first element in an ECDSA signature. Only after receiving r does the adversary choose the message m to be signed, and then obtains the signature (r, σ) on m. All other aspects of the enhanced unforgeability game remain identical to the standard game.

Evidence for enhanced unforgeability. To support our assumption that ECDSA is *enhanced* existentially unforgeable, we show that enhanced existential unforgeability holds in the following idealized model: In the random oracle (for $\mathcal{F}$) and generic group model (for $\mathbb{G}$), unconditionally.[7] See Appendix E.1.

The aforementioned result is shown via reduction. 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.

2.7 Non-Interactive Zero-Knowledge

Our protocol extensively uses NIZKs by compiling three-move zero-knowledge protocols (also known as sigma-protocols) using the Fiat-Shamir transform (FS), where the Verifier's messages are computed by the Prover using a predetermined hash function. In the random oracle model, where the hash function is treated as a random oracle, the FS transform produces NIZK argument systems.

We conclude the overview of our techniques by presenting a basic version of the interactive zero-knowledge technique (the sigma protocol) we employ. While the technique is fairly standard [9, 11, 12, 37, 53], we outline it here for convenience. The analysis is somewhat intricate and not essential for understanding our signature protocol, so this section can be skipped if desired.

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

Paillier & strong-RSA. 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 $x^e = y \mod N$. Here and throughout the paper, we use the notation $\pm m$ to denote the set $\{-(m-1)/2, \ldots, (m-1)/2\}$ if m is odd and $\{-m/2-1, \ldots, m/2\}$ if m is even. Finally, before we describe the zero-knowledge technique, we (informally) define Pedersen commitments.

Definition 2.2 (Pedersen, informal)**.** Let N be an RSA modulus and let s, $t \in \mathbb{Z}_N^*$ be non-trivial quadratic residues. A 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:

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

In words, the Prover must show that the Paillier plaintext α encrypted as C_0 is equal to α' in the Pedersen commitment $\hat{C}$, and that it lies in the range $\pm 2^\ell$ where 2^ℓ is much smaller than either $N_0, \hat{N}$. It is assumed that the Paillier modulus N_0 was generated by the Prover and the Pedersen parameters $(\hat{N}, s, t)$ were generated by the Verifier. We further assume that N_0 and $\hat{N}$ were generated as products of suitable[8] primes and that s and t are non-trivial quadratic residues in $\mathbb{Z}_{\hat{N}}^*$. 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.[9]

We now turn to the description of the NIZK for the relation R under its interactive variant (the actual proof is compiled to be non-interactive using the Fiat-Shamir transform). We perform the following Schnorr-style proof: the Prover encrypts a random value γ as $D_0 = (1+N_0)^\gamma \rho^{N_0} \mod N_0^2$ for a suitable random ρ, computes a Pedersen commitment $\hat{D} = s^\gamma t^\delta \mod \hat{N}$ to γ using a suitable random δ, and sends $(D_0, \hat{D})$ 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 $\mathbb{Z}_{\hat{N}}^*$, which was secretly generated by the Verifier and therefore would violate the strong-RSA assumption. In more detail:

1. The Prover samples elements $\gamma \leftarrow \pm 2^{\ell+\varepsilon}$, $\delta \leftarrow \pm \hat{N} \cdot 2^\varepsilon$ and $\rho \leftarrow \mathbb{Z}_{N_0}^*$ and sends $(D_0, \hat{D})$ where:

$$\begin{cases} D_0 = (1+N_0)^\gamma \rho^{N_0} \mod N_0^2 \\ \hat{D} = s^\gamma t^\delta \mod \hat{N} \end{cases}$$

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

3. The Prover sends (z_1, z_2, w) to the Verifier s.t.

$$\begin{cases} z_1 = \gamma + e\alpha \\ z_2 = \delta + e\beta \\ w = \rho \cdot r^e \mod N_0 \end{cases}.$$

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

[8] $\hat{N}$ should be obtained as products of safe primes.

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

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 NIZK (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 [12, 37, 53]. Following the standard paradigm, we show soundness via so-called *special soundness* by extracting the secrets from two accepting transcripts of the form $(D_0, \hat{D}, e, z_1, z_2, w)$ and $(D_0, \hat{D}, e', z_1', z_2', w')$ 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_0 cf. Fact D.2). Thus, the soundness-proof boils down to showing that Δ_e divides both Δ_{z_1} and 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 Pedersen parameter).[10]

To elaborate further, it is assumed that $\mathcal{S}$ knows λ such that $t^\lambda = s \mod \hat{N}$ and that λ is sampled from $[\hat{N}/2]$. Thus, without getting too deep into the details, if $\Delta_e \not| \Delta_z = \lambda\Delta_{z_1} + \Delta_{z_2}$, then $\mathcal{S}$ can solve the strong-RSA challenge by computing Euclid's extended algorithm on Δ_e and Δ_z. On the other hand, if $\Delta_e \not| \Delta_{z_1}$ or Δ_{z_2}, we claim that $\Delta_e \mid \Delta_z$ with probability at most $1/2$. To see why, observe that there exists another equaly likely $\lambda' \neq \lambda$ in $[\hat{N}/2]$ such that $t^\lambda = t^{\lambda'} = s \mod \hat{N}$, because t has order $\varphi(\hat{N})/4 \approx \frac{1}{2} \cdot (\hat{N}/2)$ and λ was sampled uniformly in $[\hat{N}/2]$. Since the Prover cannot distinguish between the two λ's given $\hat{C} = t^\lambda = t^{\lambda'}$ (in a statistical information-theoretic sense), if $\Delta_e \not| \Delta_{z_1}$ 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 $\mathcal{S}$ sampled). 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 NIZK. There is a somewhat standard way [9, 10, 11] to tweak the above NIZK to obtain an unconditional extractor (that does not rely on strong-RSA or any other hardness assumption), at the expense of higher communication cost.[11] Consider the relation

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

Notice that it's the same as the previous relation except that we got rid of the Pedersen commitment. Then, by removing $\hat{D}$ 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.

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

In this work we focus on n-out-of-n multi-party signing, and do not consider the more general t-out-of-n threshold signing for $t < n$. Such a protocol can be derived 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 [38]. One proposal to extend our protocol to the t-out-of-n threshold case is outlined in Appendix F.1.

The same technique can also be applied to 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 dependent shared preprocessed data can enable an attack that reveals the secret key.

[10] Parameter t is not completely random in $\mathbb{Z}_{\hat{N}}$ since it's a quadratic residue, but this does not affect the analysis.
[11] A similar approach appears in Lindell [50], from Boudot [9] and Brickell et al. [10]

3 Preliminaries

Notation. We let $\mathbb{Z}$ and $\mathbb{N}$ denote the set of integers and natural number, respectively. Upper case bold letters $\boldsymbol{X}, \boldsymbol{S}, \ldots$ denote sets and we write $2^{\boldsymbol{X}} = \{\boldsymbol{A} \text{ s.t. } \boldsymbol{A} \subseteq \boldsymbol{X}\}$ for the power set of $\boldsymbol{X}$. Secret values are always denoted with lower case letters $(p, q, \ldots)$ and public values are *usually* denoted with upper case letters $(A, B, N, \ldots)$. Arrow-accented letters $\vec{A}, \vec{\rho}, \ldots$ denote ordered sets, i.e. tuples. Lower case bold letters $\boldsymbol{u}, \boldsymbol{v}, \ldots$ denote random variables. 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 from secret values (so we write $(N; p, q)$ instead of (N, p, q)). For $a, b \in \mathbb{N}$, we write $a \mid b$ for 'a divides b' and $a \nmid b$ for the negation. For $N \in \mathbb{N}$, let $\mathrm{QR}(N)$ denote the set of quadratic residues i.e. $\mathrm{QR}(N) = \{a^2 \bmod N \text{ s.t. } a \in \mathbb{Z}_N^*\}$. We write $\gcd : \mathbb{N}^2 \to \mathbb{N}$ for the greatest common divisor operation, and $\varphi(\cdot)$ denotes Euler's totient function (not to be confused with ϕ which denotes a group homomorphism in Appendix E).

Groups & fields. $\mathbb{G}$ denotes an elliptic-curve group and $\mathbb{F}$ is a field (typically we write $\mathbb{F}_q$ to specify that the field has q elements). We write $\mathbb{1} \in \mathbb{G}$ for the identity element in $\mathbb{G}$. Typically, $(\mathbb{G}, g, q)$ will denote the group-generator-order tuple for ECDSA. For $t \in \mathbb{Z}_N^*$, we write $\langle t \rangle = \{t^k \bmod N \text{ s.t. } k \in \mathbb{Z}\}$ for the multiplicative group generated by t. For $\ell \in \mathbb{Z}$, we let $\pm\ell$ denote the interval of integers $\{-(|\ell| - 1)/2, \ldots, 0, \ldots, (|\ell| - 1)/2\}$ if ℓ is odd and $\{-|\ell/2| + 1, \ldots, 0, \ldots, |\ell/2|\}$ otherwise.

Algorithms, polynomials & negligible functions. We use sans-serif letters ($\mathsf{enc}, \mathsf{dec}, \ldots$) or calligraphic $(\mathcal{S}, \mathcal{A}, \ldots)$ to denote algorithms. We write $x \leftarrow \boldsymbol{E}$ or $x \leftarrow \boldsymbol{e}$ for sampling x uniformly from a set $\boldsymbol{E}$ or as a sample of $\boldsymbol{e}$ respectively, and $x \leftarrow \mathcal{A}$ or $x \leftarrow \mathsf{gen}$ for sampling x according to (probabilistic) algorithms $\mathcal{A}$ or gen respectively. For $g : \mathbb{N} \mapsto \mathbb{R}$ we say that g is polynomially bounded and we write $g \in \mathsf{poly}$ if there exists $c \in \mathbb{N}$ such that $g(\kappa) \leq \kappa^c$ for all-but-finitely-many κ's. Furthermore, for $\varepsilon : \mathbb{N} \mapsto \mathbb{R}$, we write $\varepsilon \in 1/\mathsf{poly}$ if $1/\varepsilon$ is polynomially bounded (i.e. $1/\varepsilon \in \mathsf{poly}$). A function $\nu : \mathbb{N} \mapsto \mathbb{R}$ is negligible if for every $\varepsilon \in 1/\mathsf{poly}$ it holds that $\nu(\kappa) \leq \varepsilon(\kappa)$ for all-but-finitely-many κ's and we write negl for the set of negligible functions.

Distribution ensembles & indistinguishability. A distribution ensemble $\{\boldsymbol{v}_\kappa\}_{\kappa \in \mathbb{N}}$ is a sequence of random variables indexed by the natural numbers. We say two ensembles $\{\boldsymbol{v}_\kappa\}$ and $\{\boldsymbol{u}_\kappa\}$ are ε-indistinguishable and we write $\{\boldsymbol{v}_\kappa\} \overset{\varepsilon}{\equiv} \{\boldsymbol{u}_\kappa\}$ if $|\Pr[\boldsymbol{D}(1^\kappa, \boldsymbol{u}_\kappa) = 1] - \Pr[\boldsymbol{D}(1^\kappa, \boldsymbol{v}_\kappa) = 1]| \leq \varepsilon(\kappa)$ for every efficient distinguisher $\boldsymbol{D}$, for all-but-finitely-many κ's. Finally, we write $\mathrm{SD}(\boldsymbol{u}, \boldsymbol{v})$ for the statistical distance of $\boldsymbol{u}$ and $\boldsymbol{v}$, i.e.

$$\mathrm{SD}(\boldsymbol{u}, \boldsymbol{v}) = \max_{\boldsymbol{W} \subseteq \mathrm{supp}(\boldsymbol{u})} |\Pr[\boldsymbol{v} \in \boldsymbol{W}] - \Pr[\boldsymbol{u} \in \boldsymbol{W}]|$$

3.1 Definitions

Definition 3.1. We say that $N \in \mathbb{N}$ is a Paillier-Blum integer iff $\gcd(N, \varphi(N)) = 1$ and $N = pq$ where p, q are primes such that $p, q \equiv 3 \mod 4$. For security parameter κ, we write $(N, n_{\mathsf{rsa}}; p, q) \leftarrow \mathsf{sampleRSA}(1^\kappa)$ for an algorithm that returns a Paillier-Blum modulus N with its factorization, consisting of two primes p, q of bitlength $b \in [n_{\mathsf{rsa}}, n_{\mathsf{rsa}} + 4)$ together with n_{rsa} (the minimal bit-length of N as a function of κ).

Definition 3.2 (Paillier Encryption)**.** Define the Paillier cryptosystem as the three tuple $(\mathsf{gen}, \mathsf{enc}, \mathsf{dec})$ below.

1. Let $(N, n_{\mathsf{rsa}}; p, q) \leftarrow \mathsf{sampleRSA}(1^\kappa)$. 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 $\mu = \varphi(N)^{-1} \mod N$,

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

Definition 3.3 (Pedersen Tuple)**.** Define $\mathsf{samplePed}(1^\kappa)$ to be the following parameters generating algorithm:

1. Let n_{rsa} denote the RSA bit-length returned by $\mathsf{sampleRSA}(1^\kappa)$

2. Sample random *safe primes* p, q of bit length $b \in [n_{\mathsf{rsa}}, n_{\mathsf{rsa}}+4)$ s.t. $(p-1)/2$, $(q-1)/2$ are also prime.
3. Set $\hat{N} = pq$, sample $t \leftarrow \mathrm{QR}(\hat{N})$ and $\lambda \leftarrow [\varphi(\hat{N})/4]$.
4. Return $(\hat{N}, s, t; \lambda)$ for $s = t^{\lambda} \mod \hat{N}$.

Definition 3.4 (ECDSA)**.** Let $(\mathbb{G}, g, q)$ denote the group-generator-order tuple associated with a given ECDSA 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, denoted $a_x = a|_{(\cdot)}$.

Parameters: Group-generating algorithm $\mathsf{group}(\cdot)$ and keyed hash-function family $\boldsymbol{F}_\kappa$.

0. Return the group-order-generator tuple $(\mathbb{G}, g, q) \leftarrow \mathsf{group}(1^\kappa)$ and hash function $\mathcal{F} \leftarrow \boldsymbol{F}_\kappa$.
1. $(X; x) \leftarrow \mathsf{gen}(\mathbb{G}, g, q)$ such that $x \leftarrow \mathbb{F}_q$ and $X = g^x$.
2. For $\mathrm{msg} \in \boldsymbol{M}$, let $\mathsf{sign}_x(m; k) = (r, \gamma^{-1}(m + rx)) \in \mathbb{F}_q^2$, where $\gamma \leftarrow \mathbb{F}_q^*$ and $m = \mathcal{F}(\mathrm{msg})$ and $r = g^\gamma|_{(\cdot)}$.
3. For $(r, \sigma) \in \mathbb{F}_q^2$, define $\mathsf{vrfy}_X(m, \sigma) = 1$ iff $r = (g^m \cdot X^r)^{\sigma^{-1}}|_{(\cdot)}$.

3.2 NP-relations

Schnorr. For group-generator-order tuple $(\mathbb{G}, g, q)$, the following relation verifies that the prover knows the exponent of the group element $X \in \mathbb{G}$ with respect to g. For PUB_0 of the form $(\mathbb{G}, g, q)$, define

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

Dlog with El-Gamal commitment. For public parameter $(\mathbb{G}, g, q)$, the following relation verifies that the discrete log of Y base h is equal to the discrete log base g of the El-Gamal plaintext associated with ciphertext (L, M) and public key X. For PUB_0 of the form $(\mathbb{G}, g)$, define

$$\boldsymbol{R}_{\mathsf{elog}} = \{(\mathsf{PUB}_0, X, h, A, B, Y; x, \alpha) \mid X = h^x \wedge A = g^\alpha \wedge B = g^x \cdot Y^\alpha\}.$$

Paillier encryption in range with El-Gamal commitment. For parameters $(\mathbb{G}, g, q, N_0)$ consisting of the group-generator-order tuple and the Paillier public key N_0, the following relation verifies that the plaintext value of Paillier ciphertext C is equal to the secret value of El-Gamal commitment $(A, Y, X) \in \mathbb{G}^3$ in range $\boldsymbol{I}$. For PUB_1 of the form $(\mathbb{G}, g, q, N_0)$, Define

$$\boldsymbol{R}_{\mathsf{enc\text{-}elg}} = \left\{(\mathsf{PUB}_1, \boldsymbol{I}, C, Y, A, X; x, \rho, a) \mid x \in \boldsymbol{I} \wedge C = (1+N_0)^x \rho^{N_0} \in \mathbb{Z}_{N_0^2}^* \wedge A = g^a \wedge X = g^x \cdot Y^a\right\}.$$

Paillier affine operation in range with group commitment. For parameter $(\mathbb{G}, g, q, N_0, N_1)$ consisting of the group-generator-order tuple and the 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 K such that the multiplicative coefficient (i.e. x) is equal to the exponent of $X \in \mathbb{G}$ in the range $\boldsymbol{I}$, and the additive coefficient (i.e. y) is equal to the plaintext value of $F \in \mathbb{Z}_{N_1}$ and resides in the range $\boldsymbol{J}$. For PUB_2 of the form $(\mathbb{G}, g, q, N_0, N_1)$, define $\boldsymbol{R}_{\mathsf{aff\text{-}g}}$ to be all tuples $(\mathsf{PUB}_2, \boldsymbol{I}, \boldsymbol{J}, D, K, F, X; x, y, r, \rho)$ such that

$$\begin{aligned}(x, y) \in \boldsymbol{I} \times \boldsymbol{J} \wedge D = K^x \cdot (1+N_0)^y r^{N_0} \in \mathbb{Z}_{N_0^2}^* \\ \wedge F = (1+N_1)^y \rho^{N_1} \in \mathbb{Z}_{N_1^2}^* \wedge X = g^x \in \mathbb{G}\end{aligned}$$

3.2.1 Auxiliary Relations

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

$$\begin{aligned}\boldsymbol{R}_{\mathsf{mod}} = \big\{(N; p, q) \mid p, q \in \mathsf{PRIMES} \wedge p, q \equiv 3 \mod 4 \\ \wedge N = pq \wedge \gcd(N, \varphi(N)) = 1\big\}.\end{aligned}$$

Small-factor modulus. This relation verifies that a modulus N can be factored into two 'small' numbers.

$$\boldsymbol{R}_{\mathsf{fac}} = \left\{(\ell, N; p, q) \mid N = pq \ \wedge \ p, q \in \pm\sqrt{N} \cdot 2^{\ell}\right\}.$$

Pedersen parameters. This relation verifies that $s \in \mathbb{Z}_N^*$ belongs to the group generated by $t \in \mathbb{Z}_N^*$.

$$\boldsymbol{R}_{\mathsf{prm}} = \left\{(N, s, t; \lambda) \mid s = t^{\lambda} \mod N \wedge t \in \mathbb{Z}_N^*\right\}.$$

3.2.2 Relations for Accountability

Paillier special decryption in the exponent. For parameter $(\mathbb{G}, g, q, N_0)$ consisting of the group-generator-order tuple and the Paillier public keys N_0, the following relation verifies that the plaintext-value of the Paillier ciphertext $K^x \cdot D \in \mathbb{Z}_{N_0^2}^*$, where x is the discrete log of X with respect to g, is equal to the discrete logarithm of S with respect to h. Letting $\mathsf{PUB}_1 = (\mathbb{G}, g, q, N_0)$, define $\boldsymbol{R}_{\mathsf{dec}}$ to consist of all tuples of the form $(\mathsf{PUB}_1, \boldsymbol{I}, \boldsymbol{J}, K, X, D, S, h; z, x, \rho)$ such that

$$(h^z, g^x) = (S, X) \in \mathbb{G}^2 \ \wedge \ (1 + N_0)^z \cdot \rho^N = K^x \cdot D \in \mathbb{Z}_{N_0^2}^* \wedge (x, z) \in \boldsymbol{I} \times \boldsymbol{J}.$$

Remark 3.5. In what follows, to alleviate notation when no confusion arises, we omit writing the public parameters described by PUB_*.

3.3 Cryptographic Tools

Let sRSA, DDH and DCR denote the following distributions. $(N, C) \leftarrow \mathsf{sRSA}(1^\kappa)$ for $(N, \ldots) \leftarrow \mathsf{samplePed}(1^\kappa)$ and $C \leftarrow \mathbb{Z}_N^*$, $(g^a, g^b, g^{ab+cz}, z) \leftarrow \mathsf{DDH}(1^\kappa)$ for $z \leftarrow \{0,1\}$ and $a, b, c \leftarrow \mathbb{F}_q$ and $(\mathbb{G}, g, q) \leftarrow \mathsf{group}(1^\kappa)$ for a group-generating algorith $\mathsf{group}(\cdot)$, and $(N, (1+N)^z \cdot \rho^N \bmod N^2, z) \leftarrow \mathsf{DCR}(1^\kappa)$ for $(N, \ldots) \leftarrow \mathsf{sampleRSA}(1^\kappa)$, $z \leftarrow \{0, m\}$ and $\rho, m \leftarrow \mathbb{Z}_N^*$.

Definition 3.6. We say that strong RSA, DDH and DCR hold true if for every PPTM $\mathcal{A}$ there exists $\nu \in \mathsf{negl}$ such that the probability of following events is upper-bounded by $\nu(\kappa)$, $1/2+\nu(\kappa)$, and $1/2+\nu(\kappa)$, respectively.

1. $(N, C) \leftarrow \mathsf{sRSA}(1^\kappa)$ and $(m, e \notin \{-1, 1\}) \leftarrow \mathcal{A}(1^\kappa, N, C)$ such that $m^e = C \bmod N$.
2. $(A, B, C, z) \leftarrow \mathsf{DDH}(1^\kappa)$ and $\beta \leftarrow \mathcal{A}(1^\kappa, A, B, C)$ such that $z = \beta$.
3. $(N, C, z) \leftarrow \mathsf{DCR}(1^\kappa)$ and $\beta \leftarrow \mathcal{A}(1^\kappa, N, C)$ such that $z = \beta$

FIGURE 2 (ECDSA Enhanced Unforgeability Experiment $\mathsf{EnhancedECDSA}(\mathcal{A}, d, 1^\kappa)$)

0. Compute the group-order-generator tuple $(\mathbb{G}, q, g) \leftarrow \mathsf{group}(1^\kappa)$ and hash function $\mathcal{F} \leftarrow \boldsymbol{F}_\kappa$.
1. Generate a key-pair $(x \leftarrow \mathbb{F}_q, X = g^x)$.
2. Sample $\vec{R}_i = (R_{i,j} = g^{k_{i,j}^{-1}} \leftarrow \mathbb{G})_{j \le d}$.
3. For $i = 1 \ldots d(\kappa)$
   (a) Choose $m_i \leftarrow \mathcal{A}(\mathbb{G}, g, q, \mathcal{F}, X, \vec{R}_i, m_1, \sigma_1, \ldots, m_{i-1}, \sigma_{i-1})$
   (b) Compute $\sigma_i = \mathsf{sign}_x(m_i; k_i)$.
4. $\mathcal{A}$ outputs (m, σ) on input $(\mathbb{G}, g, q, \mathcal{F}, X, \vec{R}_i, m_1, \sigma_1, \ldots, m_d, \sigma_d)$.

- **Output:** $\mathsf{EnhancedECDSA}(\mathcal{A}, d, 1^\kappa) = 1$ if $\mathsf{vrfy}_X(m, \sigma) = 1$ and $m \notin \{m_i\}_i$ and 0 otherwise.

Figure 2: ECDSA Enhanced Unforgeability Experiment $\mathsf{EnhancedECDSA}(\mathcal{A}, d, 1^\kappa)$

3.3.1 Enhanced Existential Unforgeability of ECDSA

Definition 3.7 (Enhanced Unforgeability)**.** We say that a signature scheme $(\mathsf{gen}, \mathsf{sign}, \mathsf{vrfy})$ is *enhanced* existentially unforgeable if there exists a negligible function $\nu(\cdot)$ such that for every $\mathcal{A}$ and every $d \in \mathsf{poly}$ it holds that $\Pr[\mathsf{EnhancedECDSA}(\mathcal{A}, d, 1^\kappa) = 1] \leq \nu(\kappa)$, where $\mathsf{EnhancedECDSA}(\cdot)$ denotes the enhanced unforgeability game from Figure 2.

3.4 Sigma-Protocols

In this section, we define zero-knowledge protocols with a focus on interactive three-move protocols, known as sigma-protocols. In Section 3.4.1, we compile these protocols using the random oracle via the Fiat-Shamir heuristic to generate non-interactive proofs. We define two notions of sigma-protocols. For security, we use the standard notions of completeness, honest-verifier zero-knowledge (HVZK), and soundness, which will be used in the security analysis of the protocol. Additionally, we define the notion of 'special' soundness, which allows for the extraction of the witness from two suitable protocol transcripts. This notion will also be useful in the later security analysis.

Definition 3.8. A sigma-protocol Π for relation $\boldsymbol{R}$ is a tuple $(\mathcal{P}_1, \mathcal{P}_2, \mathcal{V}_1, \mathcal{V}_2)$ of PPT algorithms such that

- $\mathcal{P}_1$ takes input $\kappa = |x|$ and random input τ and outputs A, and $\mathcal{V}_1$ outputs its random input e.
- $\mathcal{P}_2$ takes input (x, w, τ, e) and outputs z, and $\mathcal{V}_2$ takes (x, A, e, z) and outputs a bit b deterministically.

Security properties:

- *Completeness.* If $(x, w) \in \boldsymbol{R}$ then with overwhelming probability over the choice of $e \leftarrow \mathcal{V}_1$, $A \leftarrow \mathcal{P}_1(\tau)$ and $z \leftarrow \mathcal{P}_2(x, w, \tau, e)$, it holds that $\mathcal{V}_2(x, A, e, z) = 1$.
- *Soundness.* If x is false with respect to $\boldsymbol{R}$ (i.e $(x, w) \notin \boldsymbol{R}$ for all w), then for any PPT algorithm $\mathcal{P}^*$ and every $\tau^* \in \{0,1\}^*$ and A, the following holds with overwhelming probability over $e \leftarrow \mathcal{V}_1$ (as a function of κ): If $z \leftarrow \mathcal{P}^*(x, \tau^*, A, e)$ then $\mathcal{V}_2(x, A, e, z) = 0$.
- *HVZK.* There exists a simulator $\mathcal{S}$ such that $(A, e, z) \leftarrow \mathcal{S}(x)$ and $\mathcal{V}_2(x, A, e, z) = 1$ for every x, with overwhelming probability over the random coins of $\mathcal{S}$. Furthermore, the following distributions are statistically indistinguishable. For $(x, w) \in \boldsymbol{R}$:
  - (A, e, z) where $e \leftarrow \mathcal{V}_1$ and $A \leftarrow \mathcal{P}_1(x, w, \tau)$, and $z = \mathcal{P}_2(x, w, \tau, e)$.
  - (A, e, z) where $e \leftarrow \mathcal{V}_1$ and $(A, z) \leftarrow \mathcal{S}(x, e)$.

We use the above notion of sigma-protocol for proving the well-formedness of all but three of the NP-relations in Section 3.2, namely all but $\boldsymbol{R}_{\mathsf{fac}}$, $\boldsymbol{R}_{\mathsf{enc\text{-}elg}}$, and $\boldsymbol{R}_{\mathsf{aff\text{-}g}}$. For these three remaining relations, we use a variant of the sigma-protocol where all algorithms take a setup parameter as input (see the definition below). The setup parameter in Definition 3.9 is assumed to be generated by an algorithm $\mathcal{S}$. As specified in the definition, the algorithm $\mathcal{S}$ is only trusted for the property of special soundness, not for completeness or HVZK. By instructing the verifier to run $\mathcal{S}$, the resulting sigma-protocol is secure without additional trust assumptions. We further comment on the specific format of the setup parameter in Remark 3.10.

Definition 3.9. A sigma-protocol Π_σ with setup σ and special soundness for relation $\boldsymbol{R}$ is a tuple $(\mathcal{S}, \mathcal{P}_1, \mathcal{P}_2, \mathcal{V}_1, \mathcal{V}_2)$ of PPT algorithms satisfying the same functionalities and security properties of the sigma-protocol definition (w/o setup and special soundness), with the following changes:

1. Setup algorithm $\mathcal{S}$ initially generates σ which is a common input to all other algorithms.
2. Compl. & HVZK are adapted to hold with respect to a fixed σ.
3. Soundness property is replaced with:

- *Special Soundness.* There exists an efficient extractor $\mathcal{E}$ such that for any x, $\mathcal{P}^*$ and $\tau^* \in \{0,1\}^*$ the following holds with overwhelming probability over the choice of $\sigma \leftarrow \mathcal{S}$: If (A, e, z), $(A, e', z') \leftarrow \mathcal{P}^*(x, \tau^*, \sigma)$ such that $\mathcal{V}_2(x, A, e, z) = \mathcal{V}_2(x, A, e', z') = 1$ and $e \neq e'$, then for $w' \leftarrow \mathcal{E}(x, A, e, e', z, z')$ it holds that $(x, w') \in \boldsymbol{R}$.

Remark 3.10. Our main sigma-protocols for $\boldsymbol{R}_{\mathsf{fac}}$, $\boldsymbol{R}_{\mathsf{enc\text{-}elg}}$, and $\boldsymbol{R}_{\mathsf{aff\text{-}g}}$ (see Appendices A.2 to A.4) require a setup parameter $\sigma = (\hat{N}, s, t, \psi_{\mathsf{prm}})$. This consists of an RSA modulus $\hat{N}$, Pedersen parameters $s, t \in \mathbb{Z}^*_{\hat{N}}$, and a non-interactive zero-knowledge proof ψ_{prm} validating that $(\hat{N}, s, t) \in \boldsymbol{R}_{\mathsf{prm}}$ (see Section 3.4.1). We denote the respective protocols as $\Pi^{\mathsf{fac}}_{\sigma}$, $\Pi^{\mathsf{enc\text{-}elg}}_{\sigma}$, and $\Pi^{\mathsf{aff\text{-}g}}_{\sigma}$.

Notation 3.11. In our protocol, the setup parameter is generated by the Verifier (the intended recipient of the proof). The Prover generates distinct proofs (one for each Verifier using their personal σ) to prove the same statement x to multiple verifying parties. Therefore, in the sequel, we incorporate the setup parameter σ in the protocol description. We write Π^*_j for the corresponding protocol using $\mathcal{P}_j$'s setup parameter (acting as the Verifier) for $* \in \{\mathsf{fac}, \mathsf{enc\text{-}elg}, \mathsf{aff\text{-}g}\}$, and omit mentioning the algorithm $\mathcal{S}$.

3.4.1 ZK-Module

Next, we present how to compile the protocols above using a random oracle via the Fiat-Shamir heuristic. To generate a proof, the Prover computes the challenge e by querying the oracle on a suitable input, which incorporates the theorem-statement x and the first message A. The Prover then completes the transcript by computing the last message with respect to e and communicates the entire transcript as the proof. The Verifier later accepts the proof if it is a valid transcript of the underlying sigma-protocol and e is well-formed (verified by querying the oracle as the Prover did).

Formally, we define the compiler via the ZK-Module from Figure 3. Notice that, in addition to the standard prove/verify operations, the ZK-Module contains a 'commit' operation for generating the first message $A \leftarrow \mathcal{P}_1$ of the ZK-Proof. This will be useful for the signature protocol later, as the parties will be instructed to generate A ahead of time, forcing the adversary to commit to the first message of the (future) proof. This enables efficient extraction in the security analysis without relying on the forking lemma.

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

FIGURE 3 (ZK-Module $\mathcal{M}$ for sigma-protocols)

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

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

Figure 3: ZK-Module $\mathcal{M}$ for sigma-protocols

Notation 3.12. Sometimes we omit writing the randomness τ in the tuple $(\mathtt{prove}, \Pi, \mathsf{aux}, x; w, \tau)$, indicating that fresh randomness is sampled.

3.5 Communication Model and Broadcast

We assume some familiarity with the UC framework and we refer the reader for to [16] for a reminder. Communication between machines over a network is modeled using subroutine-machines that represent the behavior of the actual communication network. It is assumed that all parties are connected by point-to-point authenticated channels as well as a verifiable broadcast channel. In the protocol description, messages intended for a specific party are sent using the point-to-point channel, while messages intended for broadcast are sent over the broadcast channel. However, for simplicity, the security analysis only uses the broadcast channel,

and all messages are addressed to all parties. This convention simplifies the identification process for faulty parties by eliminating ambiguity between faulty messages (e.g. bad NIZK) and missing messages.

Communication abstraction for security analysis. In the security analysis, we assume for simplicity that the parties are connected via an authenticated, synchronous broadcast channel. This means that the computation proceeds in rounds, and each message sent by any party in a given round is made available to all parties in the next round. Formally, synchronous communication is modeled within the UC framework by $\mathcal{F}_{\text{syn}}$, the ideal synchronous communication functionality from [15, Section 7.3.3]. The broadcast property is modeled by having $\mathcal{F}_{\text{syn}}$ require that all messages are addressed to all parties.

3.5.1 Realizing 'weak' broadcast in the point-to-point model.

In many applications that forgo accountability, a weaker, non-unanimous version of the broadcast channel is implemented using an authenticated point-to-point network, where some honest parties can output a special abort symbol, meaning consensus is not always reached among all honest parties. This approach is advantageous because it adds minimal overhead in the point-to-point model. However, it sacrifices accountability, as consensus is essential for accountability, and achieving it requires full broadcast, which incurs significant overhead in the point-to-point model [26]. Additionally, consensus is crucial during key generation and key refresh, so 'weak' broadcast is not a suitable option in these cases.

Given the popularity of this technique, we outline it here for the reader's benefit. However, it's important to note that the analysis it incurs is beyond the scope of this paper; we provide this discussion solely for informational purposes.

Informal description of weak broadcast in P2P network. The basic idea for achieving 'weak' broadcast is to insert an additional round between rounds $i-1$ and i, during which each party forwards all incoming messages from round $i-1$ to the other parties. If any inconsistencies are detected, the parties are instructed to report the culprit and halt. Under these instructions, all non-aborting parties that proceed to round i are in agreement on the data they received in round i.

The aforementioned process incurs a communication complexity penalty proportional to the size of the incoming messages and a twofold increase in round complexity to account for the interlaced rounds. However, these penalties can be mitigated if (i) a collision-resistant hash function is used to compress the data, and (ii) some (or perhaps all) of the rounds can be 'piggybacked' so that the interlaced round between rounds i and $i+1$ is combined with round $i+1$. It is important to note that optimization (ii) does not preserve security generically and needs to be implemented carefully, ensuring that the messages for round $i+1$ can be sent optimistically, even before partial consensus is reached on the incoming messages in round i.

Further references and a formal discussion can be found in [43] and subsequent works.

4 Protocol Description

In this section, we present our ECDSA protocol(s), which consist of four phases. The first phase is for generating the (shared) key, which is run once (Figure 6). The second phase refreshes the secret key shares and generates the auxiliary information required for signing (Figure 7), including Paillier keys, Pedersen parameters, etc. The third phase preprocesses signatures before the messages are known (Figure 8). Finally, the fourth phase calculates and communicates the signature shares once the message to be signed is known (Figure 10).

Online vs non-interactive signing. We present two variants of our protocol, distinguished by their modes of operation: one where signature generation is performed non-interactively, and the other where it is fully online. The fully online mode of operation of the protocol is described in Figure 4, while the non-interactive mode of operation is described in Figure 5.

Remark 4.1 (Identifiers & Concurrency). We note that the protocol contains three kinds of identifiers: the session identifier *sid*, the epoch identifier *epid*, and the signing session identifier *sgid*. The session identifier *sid* is tied to the identity of the parties, the mathematical parameters, and the public key. The epoch identifier *epid* is tied to the key-refresh epoch and the auxiliary key material of the parties for that epoch (e.g., Paillier

FIGURE 4 (Threshold ECDSA: Online Signing)

- **Key-Generation:** Upon activation on input $(\texttt{keygen}, sid, i)$ do:
  1. Run the key generation phase from Figure 6 and obtain $\vec{X}_0$ and public key X.
  2. Run the auxinfo phase from Figure 7 on input $(\texttt{aux-info}, sid, \vec{X}_0, i)$ and do:
     - When obtaining output $(sid, rid, \vec{X}^*, \vec{N}, \vec{s}, \vec{t})$ and (x_i, y_i, p_i, q_i), set $epid = (sid, rid, \vec{X}^*, \vec{N}, \vec{s}, \vec{t})$
  3. Output $(\texttt{pub-key}, sid, X)$
- **Signing:** On input $(\texttt{sign}, epid, sgid, i, \text{msg})$, do:
  1. Run the pre-signing phase from Figure 8 on input $(\texttt{pre-sign}, epid, 0, i)$.
  2. Set $m = \mathcal{F}(\text{msg})$ and run the signing phase from Figure 10 on input $(\texttt{sign}, epid, sgid, 0, i, m)$.
     Obtain $(\sigma, \mathcal{Q})$, where $\mathcal{Q} = \emptyset$ if σ is a valid signature, or $\mathcal{Q} \in \boldsymbol{P}$ is the exposed corrupted party.
  3. Output $(\texttt{sig-string}, sid, sgid, \text{msg}, \sigma, \mathcal{Q})$
- **Key-Refresh:** On input $(\texttt{key-refresh}, sid, \vec{X}, i)$ for $\vec{X}$ s.t. $epid = (\ldots, \vec{X}, \ldots)$,
  1. Run the auxinfo phase from Figure 7 on input $(\texttt{aux-info}, sid, \vec{X}, i)$.
  2. Upon obtaining output $(sid, rid, i, \vec{X}^*, \vec{N}, \vec{s}, \vec{t})$ and (x_i, y_i, p_i, q_i), do:
     - Erase all presigning and auxiliary info data of the form $(epid, \ldots)$.
     - Reasign $epid = (sid, rid, \vec{X}^*, \vec{N}, \vec{s}, \vec{t})$ and

     Error. In the event of an error during key refresh, ignore all future invocations of $(\texttt{key-refresh}, \ldots)$.

Figure 4: Threshold ECDSA: Online Signing

public keys). The signing session identifier *sgid* is tied to the message being signed, and two different signature sessions for the same message are distinguished by the *sgid*. Within a specific session, identified by the *sid*, it is assumed that the key-refresh phase is executed sequentially, meaning no other protocol components for this *sid*, including another concurrent execution of the key-refresh, are executed while the key-refresh is ongoing. No other limitations are imposed on the concurrency of the other components.

FIGURE 5 (Threshold ECDSA: Non-Interactive Signing)

- **Key-Generation:** Same as in Figure 4.
- **Pre-Signing:** On input $(\texttt{pre-sign}, epid, L, i)$, do:
  1. Erase all pre-signing data $(epid, \ldots)$.
  2. Run the pre-signing phase from Figure 8 concurrently on inputs $\{(\texttt{pre-sign}, epid, \ell, i)\}_{\ell \in [L]}$
     - When obtaining output standby.
- **Signing:** On input $(\texttt{sign}, epid, sgid, \ell, i, \text{msg})$, set $m = \mathcal{F}(\text{msg})$ and do:
  1. Run the signing phase from Figure 10 on input $(\texttt{sign}, epid, sgid, \ell, i, m)$.
     Obtain $(\sigma, \mathcal{Q})$, where $\mathcal{Q} = \emptyset$ if σ is a valid signature, or $\mathcal{Q} \in \boldsymbol{P}$ is the exposed corrupted party.
  2. Output $(\texttt{sig-string}, sid, sgid, \text{msg}, \sigma, \mathcal{Q})$
- **Key-Refresh:** Same as in Figure 4.

Figure 5: Threshold ECDSA: Non-Interactive Signing

Remark 4.2 (Aborts). The protocol description instructs the parties to halt in case of an abort. However, the parties can be reactivated when they receive instructions to do so. The only assumptions we impose in such cases are (i) any reactivation uses a different identifier, and (ii) if the key-refresh resulted in an error, the key-refresh phase is not executed again. Therefore, any reattempted key generation is initiated with a new *sid*

compared to the aborted key generation, and any reattempted signature session is initiated with a new *sgid* compared to the aborted signing session. Additionally, honest parties will not participate in any reattempted key-refresh. It goes without saying that the parties are instructed to use fresh randomness in every attempted session.

Remark 4.3 (Hash Function $\mathcal{H}$). Our protocol is parametrized by a hash function $\mathcal{H}$, modelled in the security analysis as a random oracle, 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 $\mathcal{H}$ everywhere.

Nota Bene. Hereafter, let $\boldsymbol{I} = \pm 2^{\ell}$, $\boldsymbol{J} = \pm 2^{\ell'}$, $\boldsymbol{I}_{\varepsilon} = \pm 2^{\ell+\varepsilon}$ and $\boldsymbol{J}_{\varepsilon} = \pm 2^{\ell'+\varepsilon}$ denote integer intervals where ℓ, ℓ' and ε are fixed parameters (to be determined by the analysis—see Appendix C.1 for concrete parameters). We represent integers modulo N in the interval $\pm N$ (rather than using the canonical representation); this convention is crucial to the security analysis.[12]

4.1 Key Generation

FIGURE 6 (ECDSA Key-Generation)

Round 1.

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

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

Broadcast (sid, i, V_i).

Round 2.

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

Round 3.

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

   Send (sid, i, ψ_i) to all $\mathcal{P}_j$.

Output.

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

Errors. In case of failure, output the failed verification together with its index (i.e. the corrupted party), and halt.

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

Figure 6: ECDSA 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

[12] The reason is that the Paillier plaintexts are guaranteed to lie within a range that includes negative numbers. Each plaintext, say m, must thus be represented as an integer in $\pm N = \{-(N-1)/2, \ldots, (N-1)/2\}$; otherwise, even correctness may fail, as $-1 \neq N - 1 \mod q$ for $m = N - 1$ in the canonical representation. This discrepancy may also plausibly lead to attacks.

of knowledge of the exponent. If no inconsistencies are detected, the ECDSA public key is set to $X = \prod_j X_j$ and each $\mathcal{P}_i$ the public key-shares $(X_1, \ldots, X_n)$ as well as their own secret key-share x_i. See Figure 6.

Remark 4.4. The careful reader will notice that the proof for relation $\boldsymbol{R}_{\mathsf{sch}}$ is done interactively by producing the first message of the proof in round 1, and then finalizing the proof in round 3 using the unpredictable value ρ resulting from a coin-toss. This allows us to generate the proof interactively, which enables efficient extraction in the security proof. A similar technique is used for the proof of knowledge of the parties' rerandomized public key shares in Figure 7.

4.2 Key-Refresh & Auxiliary Information

At a very high level, the auxinfo and key-refresh phase proceeds as follows:

1. $\mathcal{P}_i$ samples a Paillier modulus N_i and Pedersen parameters $(\hat{N}_i, s_i, t_i)$.
2. $\mathcal{P}_i$ samples a secret sharing $(x_{i,1}, \ldots, x_{i,n})$ of 0, and computes $\vec{X}_i = (g^{x_{i,j}})_{j=1}^n$.
3. $\mathcal{P}_i$ samples ephemeral DH keys $(y_{i,1}, \ldots, y_{i,n})$, sets $\vec{Y}_i = (g^{y_{i,j}})_{j=1}^n$, and broadcasts $\vec{X}_i, \vec{Y}_i, N_i, \hat{N}_i, s_i, t_i$.
4. $\mathcal{P}_i$ sends $C_{k,i} = x_{i,k} + \mathcal{H}(sid, rid, i, Y_{k,i}^{y_{i,k}}) \mod q$ to $\mathcal{P}_k$.

In the end, each $\mathcal{P}_i$ updates their private key-share $x_i^* = x_i + \sum_{\ell=1}^n x_{\ell,i} \mod q$, as well as the public key-shares of all parties $X_j^* = X_j \cdot \prod_{\ell=1}^n X_{\ell,j}$. They also store the new tuples $(N_1, \hat{N}_i, s_1, t_1), \ldots, (N_n, \hat{N}_n, s_n, t_n)$.

ZK Proofs & Verification. For malicious security, the aforementioned process is augmented as follows:

(a) ZKP: N_j is a Paillier-Blum Modulus.

(b) ZKP: N_j can be factored into two factors no larger than $\sqrt{N} \cdot 2^{\ell+\varepsilon}$.

(c) ZKP: s_j belongs to the multiplicative group generated by t_j in $\mathbb{Z}_{\hat{N}_j}^*$.

(d) ZKP: Schnorr proof of knowledge for the discrete logarithms of $X_{j,1}, \ldots, X_{j,n}$.

(e) The tuple of group elements $(X_{j,1}, \ldots, X_{j,n})$ satisfies $\prod_{j=1}^n X_{i,j} = \mathsf{id}$.

(f) The triple $(Y_{i,j}, Y_{j,i}, C_{i,j})$ encodes the discrete logarithm of $X_{j,i}$, for all $j \neq i$.

The steps described above are interleaved to obtain the three-round protocol from Figure 7.

All of the items above are fairly intuitive. Each party is simply proving to the others that the Paillier and Pedersen parameters are well-formed and that they know the discrete logarithm of the X-group elements used to rerandomize the secret ECDSA shares.

A comment on the DH tuples. The attentive reader will notice that when rerandomizing their shares, the parties establish pairwise secret keys $\rho_{i,j}, \rho_{j,i} \in \mathbb{F}_q$ derived from the Diffie-Hellman (DH) tuples $(Y_{i,j}, Y_{j,i}, g^{y_{i,j} y_{j,i}})$. Specifically, these keys are computed as $\rho_{i,j} = \mathcal{H}(sid, rid, i, g^{y_{i,j} y_{j,i}})$ and $\rho_{j,i} = \mathcal{H}(sid, rid, j, g^{y_{i,j} y_{j,i}})$. The parties then exchange their random shifts $x_{i,j}$ (from $\mathcal{P}_i$ to $\mathcal{P}_j$) and $x_{j,i}$ (from $\mathcal{P}_j$ to $\mathcal{P}_i$) by transmitting the one-time pads $C_{j,i} = x_{i,j} + \rho_{i,j} \mod q$ and $C_{i,j} = x_{j,i} + \rho_{j,i} \mod q$, respectively. Under the DDH assumption, it is straightforward to see that the ciphertexts $C_{j,i}$ and $C_{i,j}$ do not leak any additional information to an attacker beyond what $X_{i,j}$ and $X_{j,i}$ reveal, provided that both $\mathcal{P}_i$ and $\mathcal{P}_j$ are honest. However, if either party is corrupted, both $x_{i,j}$ and $x_{j,i}$ are revealed to the attacker—an outcome that aligns with the intended purpose of these shares, as they are meant to be disclosed to the corrupted party.

In a previous version of the protocol, the $C_{i,j}$ values were calculated as Paillier encryptions using the newly obtained keys, i.e., $C_{i,j} = \mathsf{enc}_{N_i}(x_{j,i})$. This change was not motivated by a security concern, and we provide further comments in Appendix F.2.

FIGURE 7 (Auxiliary Info. & Key Refresh in Three Rounds)

Round 1.

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

(a) Sample Paillier key $(N_i, n_{\mathsf{rsa}}; p_i, q_i) \leftarrow \mathsf{sampleRSA}(1^\kappa)$.

(b) Sample Pedersen parameters $(\hat{N}_i, s_i, t_i; \lambda_i) \leftarrow \mathsf{samplePed}(1^\kappa)$
Compute $\psi_i = \mathcal{M}(\texttt{prove}, \Pi^{\mathsf{prm}}, (sid, i), (\hat{N}_i, s_i, t_i); \lambda_i)$.

(c) Sample ephemeral DH keys $y_{i,1}, \ldots, y_{i,n} \leftarrow \mathbb{F}_q$. Set $\vec{Y}_i = (Y_{i,j} = g^{y_{i,j}})_j$.

(d) Sample $x_{i,1}, \ldots, x_{i,n} \leftarrow \mathbb{F}_q$ subject to $\sum_j x_{i,j} = 0$. Set $\vec{X}_i = (X_{i,j} = g^{x_{i,j}})_j$.

(e) Sample $(A_{i,j}, \tau_j) \leftarrow \mathcal{M}(\texttt{com}, \Pi^{\mathsf{sch}})$, for $j \in \boldsymbol{P}$. Set $\vec{A}_i = (A_{i,j})_j$.

(f) Sample $rid_i, u_i \leftarrow \{0,1\}^\kappa$ and compute $V_i = \mathcal{H}(sid, i, \vec{X}_i, \vec{Y}_i, \vec{A}_i, B_i, N_i, \hat{N}_i, s_i, t_i, \psi_i, rid_i, u_i)$.

Broadcast (sid, i, V_i).

Round 2.

When obtaining (sid, j, V_j) from all $\mathcal{P}_j$, send $(sid, i, \vec{X}_i, \vec{Y}_i, \vec{A}_i, B_i, N_i, \hat{N}_i, s_i, t_i, \psi_i, rid_i, u_i)$ to all.

Round 3.

1. Upon receiving $(sid, j, \vec{X}_j, \vec{Y}_j, \vec{A}_j, B_j, N_j, \hat{N}_j, s_j, t_j, \psi_j, rid_j, u_j)$ from $\mathcal{P}_j$, do:
   - Verify $N_j, \hat{N}_j \geq 2^{n_{\mathsf{rsa}}}$ and $\prod_k X_{j,k} = \mathsf{id}_{\mathbb{G}}$ and $\mathcal{M}(\texttt{vrfy}, \Pi^{\mathsf{prm}}, (sid, j), (\hat{N}_j, s_j, t_j), \psi_j) = 1$.
   - Verify $\mathcal{H}(sid, j, \vec{X}_j, \vec{Y}_j, \vec{A}_j, B_j, N_j, \hat{N}_j, s_j, t_j, \psi_j, rid_j, u_j) = V_j$.
2. When passing above verification for all $\mathcal{P}_j$, set $rid = \oplus_j rid_j$ and do:
   - Compute $\psi'_i = \mathcal{M}(\texttt{prove}, \Pi^{\mathsf{mod}}, (sid, rid, i), N_i; (p_i, q_i))$
   - For all $j \neq i$, compute $\psi_{j,i} = \mathcal{M}(\texttt{prove}, \Pi^{\mathsf{fac}}_j, (sid, rid, i), (N_i, \kappa); p_i, q_i)$.
   - For all $j \neq i$, set $\rho_{i,j} = \mathcal{H}(sid, srid, i, Y_{j,i}^{y_{i,j}}) \in \mathbb{F}_q$ and $C_{j,i} = x_{i,j} + \rho_{i,j} \mod q$.
   - For $j \in \boldsymbol{P}$, and $\hat{\psi}_{j,i} = \mathcal{M}(\texttt{prove}, \Pi^{\mathsf{sch}}, (sid, rid, i), X_{i,j}; x_{i,j}, \tau_j)$.

   Broadcast $(sid, i, \hat{\psi}_{j,1}, \ldots, \hat{\psi}_{j,n})$ and send $(sid, i, \psi'_i, \psi_{j,i}, C_{j,i})$ to each $\mathcal{P}_j$.

Output.

1. Upon receiving $(sid, j, \psi'_j, \psi_{i,j}, C_{i,j}, \hat{\psi}_{j,1}, \ldots, \hat{\psi}_{j,n})$ from $\mathcal{P}_j$: do:
   (a) Set $\rho_{j,i} = \mathcal{H}(sid, srid, j, Y_{j,i}^{y_{i,j}})$ and $x_{j,i} = C_{i,j} - \rho_{j,i} \mod q$. Verify $g^{x_{j,i}} = X_{j,i}$.
   In case of error, broadcast $(sid, i, \mathcal{P}_j, y_{i,j})$ and halt. (decryption error)
   (b) Verify $\mathcal{M}(\texttt{vrfy}, \Pi^{\mathsf{mod}}, (sid, rid, j), N_j, \psi'_j) = 1$ and $\mathcal{M}(\texttt{vrfy}, \Pi^{\mathsf{fac}}_i, (sid, rid, j), (N_j, \kappa), \psi_{i,j}) = 1$.
   (c) For $k \in \boldsymbol{P}$, parse $\psi_{j,k} = (\hat{A}_{j,k}, \ldots)$, verify $\hat{A}_{j,k} = A_{j,k}$ and $\mathcal{M}(\texttt{vrfy}, \Pi^{\mathsf{sch}}, (sid, rid, j), X_{j,k}, \psi_{j,k}) = 1$.
2. When passing above verification for all $\mathcal{P}_j$, do:
   - Set $x_i^* = x_i + \sum_j x_{j,i} \mod q$.
   - Set $X_k^* = X_k \cdot \prod_j X_{j,k}$ for every k.

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

Errors. In case of failure, output the failed verification together with its index (i.e. the corrupted party), and halt.

When obtaining $(sid, k, \mathcal{P}_j, y_{j,k})$ from $\mathcal{P}_k$ in case of decryption error, do:

(a) Set $\rho_{k,j} = \mathcal{H}(sid, srid, k, Y_{k,j}^{y_{j,k}})$ and $x_{k,j} = C_{j,k} - \rho_{k,j} \mod q$.

(b) Check that $g^{x_{k,j}} \neq X_{k,j}$.

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

Figure 7: Auxiliary Info. & Key Refresh in Three Rounds

FIGURE 8 (ECDSA Threshold 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 $(\texttt{pre-sign}, epid, \ell, i)$, interpret $epid = (\ldots, \mathbb{G}, q, g, \boldsymbol{P}, rid, \vec{X}, \vec{N}, \vec{s}, \vec{t})$, and do:

- Sample k_i, $\gamma_i \leftarrow \mathbb{F}_q$, $\rho_i, \nu_i \leftarrow \mathbb{Z}^*_{N_i}$ and set $G_i = \mathsf{enc}_i(\gamma_i; \nu_i)$, $K_i = \mathsf{enc}_i(k_i; \rho_i)$.
- Sample $Y_i \leftarrow \mathbb{G}$ and $a_i, b_i \leftarrow \mathbb{F}_q$ and set $(A_{i,1}, A_{i,2}) = (g^{a_i}, Y_i^{a_i} g^{k_i})$, $(B_{i,1}, B_{i,2}) = (g^{b_i}, Y_i^{b_i} g^{\gamma_i})$.
- Calculate for every $j \neq i$:

$$\psi^0_{j,i} = \mathcal{M}(\texttt{prove}, \Pi^{\mathsf{enc\text{-}elg}}_j, (epid, i), (\boldsymbol{I}_\varepsilon, K_i, Y_i, A_{i,1}, A_{i,2}); (k_i, \rho_i, a_i))$$
$$\psi^1_{j,i} = \mathcal{M}(\texttt{prove}, \Pi^{\mathsf{enc\text{-}elg}}_j, (epid, i), (\boldsymbol{I}_\varepsilon, G_i, Y_i, B_{i,1}, B_{i,2}); (\gamma_i, \nu_i, b_i))$$

Broadcast $(epid, i, K_i, G_i, Y_i, A_{i,1}, A_{i,2}, B_{i,1}, B_{i,2})$ and send $(epid, i, \psi^0_{j,i}, \psi^1_{j,i})$ to each $\mathcal{P}_j$.

Round 2.

1. Upon receiving $(epid, j, K_j, G_j, Y_j, A_{j,1}, A_{j,2}, B_{j,1}, B_{j,2}, \psi^0_{i,j}, \psi^1_{i,j})$ from $\mathcal{P}_j$, do:
   - Verify $\mathcal{M}(\texttt{vrfy}, \Pi^{\mathsf{enc\text{-}elg}}_i, (epid, j), (\boldsymbol{I}_\varepsilon, K_j, Y_j, A_{j,1}, A_{j,2}), \psi^0_{i,j}) = 1$.
   - Verify $\mathcal{M}(\texttt{vrfy}, \Pi^{\mathsf{enc\text{-}elg}}_i, (epid, j), (\boldsymbol{I}_\varepsilon, G_j, Y_j, B_{j,1}, B_{j,2}), \psi^1_{i,j}) = 1$.
2. When passing above verification for all $\mathcal{P}_j$, do:
   (a) Set $\Gamma_i = g^{\gamma_i}$ and $\psi_i = \mathcal{M}(\texttt{prove}, \Pi^{\mathsf{elog}}, (epid, i), (\Gamma_i, g, B_{i,1}, B_{i,2}, Y_i); (\gamma_i, b_i))$.
   (b) For every $j \neq i$, sample $r_{i,j}, s_{i,j}, \hat{r}_{i,j}, \hat{s}_{i,j} \leftarrow \mathbb{Z}_{N_j}$, $\beta_{i,j}, \hat{\beta}_{i,j} \leftarrow \boldsymbol{J}$ 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 $\hat{F}_{j,i} = \mathsf{enc}_i(\hat{\beta}_{i,j}\,;\, \hat{r}_{i,j})$.
   - $\psi_{j,i} = \mathcal{M}(\texttt{prove}, \Pi^{\mathsf{aff\text{-}g}}_j, (epid, i), (\boldsymbol{I}_\varepsilon, \boldsymbol{J}_\varepsilon, D_{j,i}, K_j, F_{i,j}, \Gamma_i); (\gamma_i, \beta_{i,j}, s_{i,j}, r_{i,j}))$.
   - $\hat{\psi}_{j,i} = \mathcal{M}(\texttt{prove}, \Pi^{\mathsf{aff\text{-}g}}_j, (epid, i), (\boldsymbol{I}_\varepsilon, \boldsymbol{J}_\varepsilon, \hat{D}_{j,i}, K_j, \hat{F}_{i,j}, X_i); (x_i, \hat{\beta}_{i,j}, \hat{s}_{i,j}, \hat{r}_{i,j}))$.

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

Round 3.

1. Upon receiving $(epid, j, \Gamma_j, \psi_j, D_{i,j}, F_{i,j}, \hat{D}_{i,j}, \hat{F}_{i,j}, \psi_{i,j}, \hat{\psi}_{i,j})$ from $\mathcal{P}_j$, do
   - Verify $\mathcal{M}(\texttt{vrfy}, \Pi^{\mathsf{aff\text{-}g}}_i, (epid, j), (\boldsymbol{I}_\varepsilon, \boldsymbol{J}_\varepsilon, D_{i,j}, K_i, F_{j,i}, \Gamma_j), \psi_{i,j}) = 1$.
   - Verify $\mathcal{M}(\texttt{vrfy}, \Pi^{\mathsf{aff\text{-}g}}_i, (epid, j), (\boldsymbol{I}_\varepsilon, \boldsymbol{J}_\varepsilon, \hat{D}_{i,j}, K_i, \hat{F}_{j,i}, X_j), \hat{\psi}_{i,j}) = 1$.
   - Verify $\mathcal{M}(\texttt{vrfy}, \Pi^{\mathsf{elog}}, (epid, j), (\Gamma_j, g, B_{j,1}, B_{j,2}, Y_j), \psi_j) = 1$.
2. When passing above verification for all $\mathcal{P}_j$, set $\Gamma = \prod_j \Gamma_j$ and $\Delta_i = \Gamma^{k_i}$ and $S_i = \Gamma^{\chi_i}$, 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

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

   - Calculate $\psi'_i = \mathcal{M}(\texttt{prove}, \Pi^{\mathsf{elog}}, (epid, i), (\Delta_i, \Gamma, A_{i,1}, A_{i,2}, Y_i); (k_i, a_i))$.

   Send $(epid, i, \delta_i, S_i, \Delta_i, \psi'_i)$ to each $\mathcal{P}_j$.

Output.

1. Upon receiving $(epid, j, \delta_j, S_j, \Delta_j, \psi'_j)$ from $\mathcal{P}_j$, do:
   - Verify $\mathcal{M}(\texttt{vrfy}, \Pi^{\mathsf{elog}}, (epid, j), (\Delta_j, \Gamma, A_{j,1}, A_{j,2}, Y_j), \psi'_j) = 1$.
2. When passing above verification for all $\mathcal{P}_j$, set $\delta = \sum_j \delta_j$, and do:
   - Verify $g^\delta = \prod_j \Delta_j$ and $X^\delta = \prod_j S_j$. In case of failure, report a red alert and go to Figure 9.
   - Output $\left(epid, \ell, i, \Gamma, k_i/\delta, \chi_i/\delta, (\Delta_j^{\delta^{-1}}, S_j^{\delta^{-1}})_{j \in \boldsymbol{P}}\right)$.

Errors. In case of failure, output the failed verification together with its index (i.e. the corrupted party), and halt.

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

Figure 8: ECDSA Threshold Pre-Signing

4.3 Presigning

Next, we present the presigning phase (Figure 8). Recall that at the end of the auxinfo phase, each $\mathcal{P}_i$ has a Paillier encryption scheme $(\mathsf{enc}_i, \mathsf{dec}_i)$ with public key N_i, as well as Pedersen parameters $(\hat{N}_i, s_i, t_i \in \mathbb{Z}_{N_i})$. Further, recall that an ECDSA signature has the form $(r = g^{\gamma}|_{(\cdot)}, \sigma = (m + rx) \cdot \gamma^{-1})$, where $\mathcal{P}_i$ has an additive share x_i of x.

Presigning Overview. The protocol proceeds as follows (each numbered item below denotes a round):

1. $\mathcal{P}_i$ generates local shares k_i and γ_i, computes Paillier ciphertexts $K_i = \mathsf{enc}_i(k_i)$ and $G_i = \mathsf{enc}_i(\gamma_i)$, under its own key, and broadcasts (K_i, G_i).

2. For each $j \neq i$, party $\mathcal{P}_i$ samples $\beta_{i,j}, \hat{\beta}_{i,j} \leftarrow \boldsymbol{J}$ 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 $(\Gamma_i, D_{j,i}, \hat{D}_{j,i}, F_{j,i}, \hat{F}_{j,i})$ to party $\mathcal{P}_j$.

3. $\mathcal{P}_i$ 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 $\Gamma = \prod_j \Gamma_j$, $\Delta_i = \Gamma^{k_i}$, $S_i = \Gamma^{\chi_i}$ and sends δ_i, Δ_i, S_i to all parties.

When obtaining all $(\delta_j, \Delta_j, S_j)_{j \neq i}$, $\mathcal{P}_i$ sets $\delta = \sum_j \delta_j \mod q$ and verifies that $g^{\delta} = \prod_j \Delta_j$ and $X^{\delta} = \prod_j S_j$. If no inconsistencies are detected, $\mathcal{P}_i$ stores $(\Gamma, k_i/\delta, \chi_i/\delta)$. To ensure accountability in the subsequent signing phase, each party additionally stores the tuple $(\tilde{\Delta}_j, \tilde{S}_j)_{j \in \boldsymbol{P}}$, where $\tilde{\Delta}_j = \Delta_j^{\delta^{-1}}$ and $\tilde{S}_j = S_j^{\delta^{-1}}$ serve as public commitment values for $\mathcal{P}_j$'s (future) signature share.

ZK Proofs. For malicious security, the aforementioned process is augmented with the following ZK-proofs:

(a) The plaintexts of K_i, G_i lie in range $\boldsymbol{I}_{\varepsilon}$.

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

(c) The discrete logarithm of Δ_i with respect to Γ is equal to the plaintext-value of K_i.

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

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 these values are encrypted under the malicious parties' Paillier keys in K_i, G_i, and $\{F_{i,j}, \hat{F}_{i,j}\}_{j \neq i}$, respectively, and the Paillier keys were extracted in the simulation of the auxiliary info phase, we can extract the desired values without issue.

4.3.1 Accountability during Presigning

If $\prod_j \Delta_j \neq g^{\delta}$, then $\mathcal{P}_i$ follows the process from Figure 9. Analogously, if $\prod S_j \neq X^{\delta}$, then $\mathcal{P}_i$ proves in zero-knowledge that $(\boldsymbol{I}_{\varepsilon}, \boldsymbol{J}_{\varepsilon}, \hat{D}_{j,i}, K_j, \hat{F}_{j,i}, X_i) \in R_{\mathsf{aff\text{-}g}}$ for all $j \in \boldsymbol{P}$ and $(\boldsymbol{I}_{\varepsilon}, \boldsymbol{J}_{\varepsilon}, K_i, X_i, \hat{D}_i, S_i, \Gamma) \in R_{\mathsf{dec}}$ for $\hat{D}_i = \prod_j \hat{D}_{i,j} \cdot \hat{F}_{j,i} \mod N_i^2$. To achieve this, the setupless statistically sound protocols $\Pi^{\mathsf{aff\text{-}g}*}$ from Appendix A.5 and Π^{dec} from Appendix A.6 are used.

4.4 Signing

Once $m = \mathcal{H}(\text{msg})$ is known, signing boils down to the following process: Each $\mathcal{P}_i$ retrieves $(\Gamma, \tilde{k}_i, \tilde{\chi}_i)$ computes $r = \Gamma|_{(\cdot)}$ and sends $\sigma_i = \tilde{k}m + r\tilde{\chi} \mod q$ to all. The tuple $(\Gamma, \tilde{k}_i, \tilde{\chi}_i)$ is erased once it is consumed. For accountability, using the supplementary presigning material $(\tilde{\Delta}_j, \tilde{S}_j)_{j \in \boldsymbol{P}}$, it is immediate that (r, σ) is a valid signature for $\sigma = \sum_j \sigma_j$, as long as $\Gamma^{\sigma_j} = \tilde{\Delta}_j^m \cdot \tilde{S}_j^r$, for all $j \in \boldsymbol{P}$. See Figure 10 for further details.

FIGURE 9 (Failed Nonce or Chi)

Round 1. In case of red alert with $\prod_{j\in\boldsymbol{P}} \Delta_j \neq g^{\delta}$ (the case $\prod_{j\in\boldsymbol{P}} S_j \neq X^{\delta}$ is analogous), do:

- Set $\{D_\ell = \prod_{j\neq\ell} D_{\ell,j} \cdot F_{j,\ell} \mod N_\ell^2\}_{\ell\in\boldsymbol{P}}$ and calculate the randomizer ρ_i of $K_i^{\gamma_i} \cdot D_i \mod N_i^2$.
- Set $\psi_i^* = \mathcal{M}(\texttt{prove}, \Pi^{\mathsf{dec}}, (epid, i), (\boldsymbol{I}_\varepsilon, \boldsymbol{J}_\varepsilon, K_i, \Gamma_i, D_i, g^{\delta_i}, g); (\delta_i, \gamma_i, \rho_i))$.
- Set $\psi_{i,j} = \mathcal{M}(\texttt{prove}, \Pi^{\mathsf{aff\text{-}g*}}, (epid, i), (\boldsymbol{I}_\varepsilon, \boldsymbol{J}_\varepsilon, D_{j,i}, K_j, F_{j,i}, \Gamma_i); (\gamma_i, \beta_{i,j}, s_{i,j}, r_{i,j}))$, for all $j \neq i$.

  Broadcast $(sid, i, \psi_i^*, \psi_{i,1}, \ldots, \psi_{i,i-1}, \psi_{i,i+1}, \ldots, \psi_{i,n})$.

Output. When obtaining $(sid, j, \psi_j^*, \psi_{j,1}, \ldots, \psi_{j,j-1}, \psi_{j,j+1}, \ldots, \psi_{j,n})$ from $\mathcal{P}_j$, do

- Check $\mathcal{M}(\texttt{vrfy}, \Pi^{\mathsf{dec}}, (epid, j), (\boldsymbol{I}_\varepsilon, \boldsymbol{J}_\varepsilon, K_j, \Gamma_j, D_j, g^{\delta_j}, g), \psi_j^*) = 1$.
- For all $\ell \neq i, j$, check $\mathcal{M}(\texttt{vrfy}, \Pi^{\mathsf{aff\text{-}g*}}, (epid, j), (\boldsymbol{I}_\varepsilon, \boldsymbol{J}_\varepsilon, D_{\ell,j}, K_\ell, F_{\ell,j}, \Gamma_j), \psi_{j,\ell}) = 1$.

Errors. When failure, output the failed verification together with its index (i.e. the corrupted party), and halt.

Figure 9: Failed Nonce or Chi

FIGURE 10 (ECDSA Signing)

Round 1. On input $(\texttt{sign}, epid, \ell, i, m)$, if there is record of $(epid, \ell, \Gamma, \tilde{k}_i, \tilde{\chi}_i)$ and $(\tilde{\Delta}_j, \tilde{S}_j)_{j\in\boldsymbol{P}}$, do:

- Set $r = \Gamma|_{(\cdot)}$ and $\sigma_i = \tilde{k}_i m + r\tilde{\chi}_i \mod q$.
- Send $(epid, i, \sigma_i)$ to all $\mathcal{P}_j$.

  Erase $(epid, \ell, \Gamma, \tilde{k}_i, \tilde{\chi}_i)$ from memory.

Output. Upon receiving $(epid, j, \sigma_j)$ from all $\mathcal{P}_j$, do:

- Check that $\Gamma^{\sigma_j} = \tilde{\Delta}_j^m \cdot \tilde{S}_j^r$ for all $j \in \boldsymbol{P}$.

  Output $\sigma = \sum_j \sigma_j$.

Errors. In case of failure, output the failed verification together with its index (i.e. the corrupted party), and halt.

Figure 10: ECDSA Signing

5 Underlying Sigma-Protocols

We present the sigma-protocols associated with the NP-relations from Section 3.2, which are invoked (using the Fiat-Shamir transform) by the parties in the protocols of Section 4. Along the way, we prove the completeness, HVZK, soundness, or special soundness properties that will be useful for the overall security analysis in Section 6. The dlog-style protocols (i.e. Schnorr), as well as those very similar to the ones below, are moved to Appendix A.

5.1 Paillier Encryption in Range ZK

NP-Relation. For Paillier public key N_0, the following relation verifies that the plaintext value of Paillier ciphertext C is in a desired range $\boldsymbol{I} = \pm 2^\ell$. Define

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

In Figure 11, we present a sigma-protocol where the completeness and HVZK relations hold with respect to $\boldsymbol{R}_{\mathsf{enc}}$, but where the special soundness property (see Claim 5.1) holds with respect to the following related relation: Define $\widetilde{\boldsymbol{R}}_{\mathsf{enc}}$ to consist of all tuples $(N_0, \boldsymbol{I}_\varepsilon, C, S, \hat{N}, t, s; x, \rho, p, q, z, z', e)$ such that

$$(N_0, \boldsymbol{I}_\varepsilon, C; x, \rho) \in \boldsymbol{R}_{\mathsf{enc}} \vee (N_0, \boldsymbol{I}_\varepsilon; p, q) \notin \boldsymbol{R}_{\mathsf{fac}} \cap \boldsymbol{R}_{\mathsf{mod}} \vee \left(s^z t^{z'} = S^e \mod \hat{N} \wedge e \nmid z \right),$$

where $(\hat{N}, s, t)$ denotes the auxiliary setup parameter for the proof, and $S \in \mathbb{Z}^*_{\hat{N}}$ is generated on the fly during the proof generation.

As we shall see later in the security analysis, special soundness with respect to $\widetilde{\boldsymbol{R}}_{\mathsf{enc}}$ suffices for our purpose since $(N_0, \boldsymbol{I}_\varepsilon) \in \boldsymbol{R}_{\mathsf{fac}} \cap \boldsymbol{R}_{\mathsf{mod}}$ is validated elsewhere in the protocol (see Figure 12 and Figure 26). Additionally, $s^z t^{z'} = S^e \mod \hat{N}$ with $e \not| z$ breaks strong RSA for $(\hat{N}, s, t) \leftarrow \mathsf{samplePed}(1^\kappa)$ (see Theorem 5.2).

5.1.1 Protocol Π^{enc}

FIGURE 11 (Paillier Encryption in Range ZK—Π^{enc})

- **Setup:** Auxiliary RSA modulus $\hat{N}$ and 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

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

   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**:

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

- **Range Check**:

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

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

Figure 11: Paillier Encryption in Range ZK—Π^{enc}

Completeness. The protocol may reject a valid $(N_0, \kappa; k, \rho) \in \boldsymbol{R}_{\mathsf{enc}}$ only if $|\alpha| \notin \pm(2^{\ell+\varepsilon} - q2^\ell)$ which happens with probability at most $q/2^\varepsilon$. □

Honest Verifier Zero-Knowledge. The simulator samples $e \leftarrow \pm q$, $z_1 \leftarrow \pm 2^{\ell+\varepsilon}$, $z_2 \leftarrow \mathbb{Z}^*_{N_0}$, $z_3 \leftarrow \pm\hat{N} \cdot 2^{\ell+\varepsilon}$, and $S \leftarrow \langle t \rangle$ by setting $S = t^\lambda \mod \hat{N}$ 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 $C = s^{z_1} t^{z_3} \cdot S^{-e} \mod \hat{N}$. From Facts D.5, D.6, we deduce that z_1, z_3 are (each) $(q2^{-\varepsilon})$-close to the real distribution, and S is $(2^{-\ell})$-close to the real distribution. Thus, since e and z_2 are identically distributed with the real transcript (and all other items are determined), it follows that the real and simulated distributions are $(2 \cdot q2^{-\varepsilon} + 2^{-\ell})$-statistically close. □

5.1.2 Special Soundness of the Range Proof & Reduction to Strong RSA

Claim 5.1. *Let $(\hat{N}, s, t, N_0, K, S, A, C, e, z_1, z_2, z_3)$ and $(\hat{N}, s, t, N_0, K, S, A, C, e', z_1', z_2', z_3')$ denote two accepting transcripts for Π^{enc} such that $e \neq e'$. Letting $\Delta_x = x - x'$ for $x \in \{e, z_1, z_3\}$ and $\Delta_{z_2} = z_2 \cdot z_2'^{-1} \mod N_0$, at least one of the following holds true:*

1. $s^{\Delta_{z_1}} \cdot t^{\Delta_{z_3}} = S^{\Delta_e} \mod \hat{N}$ *and* $\Delta_e \nmid \Delta_{z_1}$.
2. $\gcd(\Delta_e, N_0) \neq 1$ *i.e.* $(N_0, \boldsymbol{I}_\varepsilon) \notin \boldsymbol{R}_{\mathsf{fac}} \cap \boldsymbol{R}_{\mathsf{mod}}$.
3. *For* $k = \Delta_{z_1}/\Delta_e \in \boldsymbol{I}_\varepsilon$, *there exists* $\rho \in \mathbb{Z}^*_{N_0^2}$ *s.t.* $K = (1+N_0)^k \cdot \rho^{N_0} \bmod N_0^2$, *i.e.* $(N_0, \boldsymbol{I}_\varepsilon, K) \in \boldsymbol{R}_{\mathsf{enc}}$.

Proof. We prove that Item 3 holds assuming Δ_e divides both Δ_{z_1} and $\gcd(\Delta_e, N_0) = 1$ (i.e. Items 1 and 2 do not hold). First, we know that $\Delta_{z_1} \in \boldsymbol{I}_\varepsilon$ and thus, for $k = \Delta_{z_1}/\Delta_e \in \boldsymbol{I}^\varepsilon$, we deduce that

$$(K \cdot (1+N_0)^{-k})^{\Delta_e} = \Delta_{z_2}^{N_0} \mod N_0^2.$$

Next, since $\gcd(N_0, \Delta_e) = 1$, by Fact D.2, there exists $\rho \in \mathbb{Z}^*_{N_0^2}$ such that $\rho^{N_0} = K \cdot (1+N_0)^{-k} \mod N_0^2$. □

Theorem 5.2 (Implicit in [27, 37, 53])**.** *Assume that*

$$\Pr_{(\hat{N},s,t)\leftarrow\mathsf{samplePed}(1^\kappa)}\left[(\alpha,\beta,e,V) \leftarrow \mathcal{A}(\hat{N},s,t) \;\; s.t. \;\; t^\alpha s^\beta = V^e \mod \hat{N} \wedge |e| < N^{1/4} \wedge e \nmid \beta\right] \geq \delta,$$

for some PPTM $\mathcal{A}$. *Then, there exists PPTM* $\mathcal{A}_0$, *such that*

$$\Pr_{(N,t)\leftarrow\mathsf{sRSA}(1^\kappa)}\left[(m,c) \leftarrow \mathcal{A}_0(N,t) \;\; s.t. \;\; m^c = t \mod \hat{N} \wedge c \notin \{-1,1\}\right] \geq \delta \cdot \left(\frac{\varphi(N)}{N-1} - \frac{1}{2}\right).$$

Proof. Define algorithm $\mathcal{A}_0(N,t)$ as follows:

1. Set $\hat{N} = N$, sample $x \leftarrow [(N-1)/2]$ and set $s = t^x \mod \hat{N}$.
2. Execute $\mathcal{A}$ on input $(\hat{N}, s, t)$; obtain (α, β, e, V). Set $\sigma = \alpha + x \cdot \beta$ and let u, v denote the Bézout coefficients of e and $-\sigma$, i.e. $ue - v\sigma = \gcd(e, \sigma)$.
3. Output $(R, \hat{R}, c) = (t^u V^{-v}, t^u (-V)^{-v}, e/\gcd(e,\sigma))$ and $A = t^{\sigma/\gcd(e,\sigma)} \cdot V^{-e/\gcd(e,\sigma)}$.

We begin by observing that if $e \nmid \sigma$, then either $g = R^c$ or $g = \hat{R}^c$ and $c \notin \{-1,1\}$, or A is a non-trivial root of 1 (which yields a suitable pair (R,c) via the factorization of $\hat{N}$). Let $e' = e/\gcd(e,\sigma) \notin \{-1,1\}$ and $\sigma' = \sigma/\gcd(e,\sigma)$. Fix ζ to be one of the two nontrivial root of 1. Since $p = q = 3 \mod 4$, we know that $-1 \notin \mathrm{QR}_p, \mathrm{QR}_q$ and there exists a unique triple $(V_0, i, j) \in \mathrm{QR}_{\hat{N}} \times \{0,1\}^2$ such that $V = V_0 \cdot (-1)^i \cdot \zeta^j$. Furthermore, since $\hat{N}$ is a strong-prime modulus and $\gcd(e,\sigma) \leq e < \hat{N}^{1/4}$, we have that $t^{\sigma'} = V_0^{e'} \mod \hat{N}$. Consequently, either (i) $A = t^{\sigma'} V^{-e'} \in \{\zeta, -\zeta\}$, or (ii) $t^{\sigma'} V^{-e'} = -1$ and e' is odd, or (iii) $t^{\sigma'} V^{-e'} = 1$. Case (i) yields a non-trivial root of 1. For Cases (ii) and (iii), we deduce that $\hat{R}^c = t$ and $R^c = t$, respectively, since

$$\begin{cases} \hat{R}^c = \hat{R}^{e'} = t^{ue'}(-V)^{-ve'} = t^{ue'} t^{-v\sigma'} = t^{ue'-v\sigma'} = t \mod \hat{N} & \text{in Case (ii).} \\ R^c = R^{e'} = t^{ue'} V^{-ve'} = t^{ue'} t^{-v\sigma'} = t^{ue'-v\sigma'} = t \mod \hat{N} & \text{in Case (iii).} \end{cases}$$

To conclude the proof, we argue that the probability that $e \mid \sigma$ is bounded above by $1/2 + \varepsilon_0$, for $\varepsilon_0 = 1 - \varphi(\hat{N})/(\hat{N}-1)$. Using the notation above, consider (inefficient) algorithm $\mathcal{B}$ which is similar to $\mathcal{A}_0$ except that (i) instead of sampling α uniformly in $[(\hat{N}-1)/2]$ as in Item 1, $\mathcal{B}$ samples $x_0 \leftarrow [\varphi(\hat{N})/4]$ and sets $s = t^{x_0}$, (ii) When obtaining (α, β, e, V) from $\mathcal{A}$, $\mathcal{B}$ samples $\mu \leftarrow \{0,1\}$ and sets $x = x_0 + \mu \cdot \varphi(\hat{N})/2$ and, (iii) all other values are set according to $\mathcal{A}_0$. Next, we consider a random execution of $\mathcal{B}$.

Let $\boldsymbol{d}_\mu$ denote the event that e divides $\alpha + (x_0 + \mu \cdot \varphi(\hat{N})/4) \cdot \beta$. Notice that if $\boldsymbol{d}_0$ and $\boldsymbol{d}_1$ both occur (or $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$), then e divides $\varphi(\hat{N})/4 \cdot \beta$, and since $\hat{N}$ is a strong-prime modulus and $e < N^{1/4}$, it follows that $e \mid \beta$. Thus, if $e \nmid \beta$ then $\Pr[e \mid \sigma] \leq \frac{1}{2}$ in a random execution of $\mathcal{B}$.

Finally, by noting that $\alpha \leftarrow \mathcal{B}$ and $\alpha \leftarrow \mathcal{A}_0$ are statistically ε_0-close, and thus the two executions are statistically ε_0-indistinguishable, we deduce that $\Pr[e \mid \sigma] \leq 1/2 + \varepsilon_0$ in a random execution of $\mathcal{A}_0$. □

FIGURE 12 (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 $y_i' = (-1)^{a_i} w^{b_i} y_i \mod N$ for unique $a_i, b_i \in \{0, 1\}$ such that x_i is well defined.
   - $z_i = y_i^{N^{-1} \mod \varphi(N)} \mod N$

   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 and $\gcd(w, N) = 1$.
  - $z_i^N = y_i \mod N$ and $y_i \in \mathbb{Z}_N^*$ 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 12: Paillier-Blum Modulus ZK—Π^{mod}

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

In Figure 12 we give a sigma-protocol for tuples $(N; p, q)$ satisfying relation $\boldsymbol{R}_{\mathsf{mod}}$. The Prover claims that N is a Paillier-Blum modulus, i.e. $\gcd(N, \varphi(N)) = 1$ 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 [62] and Goldberg et al. [42].

Completeness. Probability 1 by construction. □

Soundness. We first observe that the probability that y_i admits an N-th root if $\langle N, \varphi(N)\rangle \neq 1$ is at most $1/\langle N, \varphi(N)\rangle \leq 1/2$. Therefore, with probability 2^{-m}, it holds that $\langle N, \varphi(N)\rangle = 1$, 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 a non-square modulo N with Jacobi symbol 1, the real and simulated distributions are identical. □

5.2.1 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.4).

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 random uniform 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 a non-square modulo N with Jacobi symbol 1, and thus the y_i's are truly random, as long as w has Jacobi symbol -1. □

5.3 Pedersen Parameters ZK (Π^{prm})

The sigma-protocol of Figure 13 for the relation $\boldsymbol{R}_{\mathsf{prm}}$ is a ZK-protocol for proving that s belongs to the multiplicative group generated by t modulo N.

FIGURE 13 (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 $\{a_i \leftarrow \mathbb{Z}_{\varphi(N)}\}_{i\in[m]}$ 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 $\{z_i = a_i + e_i\lambda \mod \varphi(N)\}_{i\in[m]}$ to the Verifier.
- **Verification:** Accept if $t \in \mathbb{Z}_N^*$ and $t^{z_i} = A_i \cdot s^{e_i} \mod N$, for every $i \in [m]$.

Figure 13: Pedersen Parameters ZK—Π^{prm}

Completeness. Probability 1, by construction. □

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

Zero-Knowledge. Sample $\{z_i \leftarrow \pm N\}_{i\in[m]}$ 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 $m \cdot (1 - \varphi(N)/N)$-close. □

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

5.3.1 On the Auxilliary RSA moduli and the 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 $s \in \langle t\rangle$.

Sampling Uniform Elements 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 $\varphi(\hat{N})$ by sampling elements in $\hat{N}$. However, if $\hat{N}$ has small factors,[13] then small values close to zero will have noticeably more weight than other values, modulo $\varphi(\hat{N})$. To fix this issue, we instruct the Prover (and the simulator in the proof of the HVZK property) to sample elements from $\pm 2^{\ell} \cdot \hat{N}$. That way, modulo $\varphi(\hat{N})$, the resulting distribution is $\frac{1}{2^{\ell}}$-far from the uniform distribution in $\varphi(N)$, by Fact D.6.

Choice of Moduli. With respect to our ECDSA protocol, for the $\Pi^{\mathsf{enc\text{-}elg}}$ 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 moduli of the Verifier, i.e. the 'receiver' of the homomorphic evaluation, and N_1 is the modulus of the Prover, i.e. homomorphic 'evaluator'. See the presigning protocol from Figure 8 for further details.

6 Security Analysis

In this section, we show that our protocol UC-realizes a proactive ideal threshold signature functionality ($\mathcal{F}_{\mathsf{tsig}}$ from Figure 15). This section presumes familiarity with the UC framework (cf. [16]). We adopt the global random oracle model for our security analysis and assume that all hash values (e.g. for the Fiat-Shamir Heuristic) are obtained by querying the random oracle, defined next.

6.1 Global Random Oracle

We use the formalism of Canetti et al. [20], Camenisch et al. [13] to incorporate the random oracle model within the UC framework. This formalism accounts for the fact that the random oracle is an abstraction of an actual public hash function used globally across the analyzed system and its environment. Specifically, the random oracle is modeled as an ideal functionality that is globally accessible in both the real system and the ideal system. Canetti et al. [20], Camenisch et al. [13] provide a number of alternative formulations for the functionality representing the random oracle. Here, we use the simplest (and most restrictive) formulation, called the *strict random oracle,* which does not allow for observation of the queries or programming of the oracle responses. The incorporation of global functionalities within the plain UC model is done using [2].[14]

The functionality takes inputs of arbitrary size and is parameterized by the output space $\boldsymbol{O}$. When queried on a new value $m \in \{0,1\}^*$, the functionality returns a value uniformly chosen from $\boldsymbol{O}$. All future queries for m return the same value.

FIGURE 14 (The Global Random Oracle Functionality $\mathcal{H}$)

Parameter: Output space $\boldsymbol{O}$.

- 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 $a \leftarrow \boldsymbol{O}$ and store (m, a).

    Output $(\texttt{answer}, a)$ to $\mathcal{X}$.

Figure 14: The Global Random Oracle Functionality $\mathcal{H}$

[13] 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 $p \sim 2^{\kappa}/\mathsf{poly}(\kappa)$

[14] The fact that our analysis works even with the strict formalization of the random oracle means that it would work with any of the other (more elaborate) variants discussed in Canetti et al. [20], Camenisch et al. [13].

6.2 Ideal Threshold Signature Functionality

Next, we describe our ideal threshold signature functionality. This functionality is largely an adaptation of the (non-threshold) signature functionality from Canetti [14], with added constructs that that account for multiple signatories, for accountability, and for proactive security. See Figure 15 for the formal description of the functionality. We also comment on some of the differences between [14] and our functionality further below.

Initialization & Key generation. At first activation (by a party called $\mathcal{P}_0$, representing the administrative contact for the signature service) $\mathcal{F}_{\mathsf{tsig}}$ receives its session ID *sid* which encodes in it the set of signatories $\boldsymbol{P} = \{\mathcal{P}_1 \dots \mathcal{P}_n\}$. It also intializes a global state variable to `uncompromised` and a refresh counter to $c = 0$. When prompted by all parties to generate a public key, $\mathcal{F}_{\mathsf{tsig}}$ obtains a verification key (effectively, a verification algorithm) $\mathcal{V}$ from the adversary and returns this key to the administrative owner.

Signature generation. When all signatories prompt the functionality to obtain a signature for some message m and signature ID *sgid*, the functionality requests a string σ from the adversary. Now, if σ verifies (i.e., if $\mathcal{V}(m, \sigma) = 1$) then $\mathcal{F}_{\mathsf{tsig}}$ records $(sgid, m, \sigma, 1)$ and outputs σ to $\mathcal{P}_0$. Else, it outputs the identity of one of the corrupted parties (chosen by the adversary) as the 'culprit' for the generation of a bad signature.

This last provision captures the 'identifiable abort' guarantee: a failed signature generation is bound to identify at least one corrupted party.

Signature verification. When asked to verify some signature σ for a message m and public key $\mathcal{V}'$, the functionality proceeds as follows. If (i) $\mathcal{V}' = \mathcal{V}$, (ii) the global state is `uncompromised` and (iii) there exists a party that did not ask to sign m, then a 'fail' response is returned. In all other cases, the value $\mathcal{V}'(m, \sigma)$ is returned—since $\mathcal{V}$ is treated by the functionality as a deterministic algorithm, consistency across multiple verification requests of the same signature is guaranteed.

It is noted that the unforgeability guarantee provided by the above logic is unconditional, and remains the case even for unbounded environments. Furthermore, the response to the verification request is generated within a single activation of the functionality, without invoking the adversary (which could create delays in generating the response). This represents the requirement that signature verification must be a local process which can provide an immediate response.

Modeling corruption. When instructed by the adversary to corrupt party $\mathcal{P}_i$ the functionality records $\mathcal{P}_i$ as `corrupted`. When instructed by the adversary to uncorrupt $\mathcal{P}_i$, the functionality records $\mathcal{P}_i$ as `quarantined`. If at any point in time all parties are either corrupted or quarantined, the global state is set to `compromised`. Once compromised, the state does not change any more. When a party queries for its corruption status, the status is returned.[15]

Proactive key refresh. When instructed by a party to initiate a key refresh, the functionality informs the adversary of the request. When the adversary notifies the functionality that a key refresh has been completed, it updates the state of all the `quarantined` parties to `uncorrupted` and increments the refresh counter c. Whenever some $\mathcal{P}_i$ asks for the value of c, the value is returned.

6.2.1 Discussion

We discuss some modeling choices and differences from the ideal signature functionality of [14].

Recording of signature strings. In [14], the functionality stores all signature strings, including invalid ones—ie strings that are rejected by the functionality. This ensures consistency—namely that invalid signatures remain invalid and valid signatures remain valid throughout the execution. In our case, we do not need to record past signatures because (a) unforgeability is guaranteed by making sure that only messages that all parties asked to sign would ever be accepted as signed, and (b) consistency is guaranteed through the verification

[15] This last stipulation prevents the undesirable situation where the ideal model adversary surreptitiously corrupts parties even when these parties were not corrupted by the environment.

FIGURE 15 (Ideal Threshold Signature Functionality $\mathcal{F}_{\mathsf{tsig}}$)

Initialization:

At first activation by party $\mathcal{P}_0$, interpret the given session ID as a tuple $sid = (\ldots, \boldsymbol{P})$, where $\boldsymbol{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ is a set of *signatories*. Record all signatories as `non-corrupted`, and initialize a refresh counter $c = 0$ and a state variable to `uncompromised`.

Key-generation:

1. Upon receiving (`keygen`) from some party $\mathcal{P}_i$ where $\mathcal{P}_i \in \boldsymbol{P}$, send (`keygen`, i) to $\mathcal{S}$.
2. Upon receiving (`pub-key`, $\mathcal{V}$) from $\mathcal{S}$, if all $\mathcal{P}_i \in \boldsymbol{P}$ have already input `keygen`, and no tuple (`pub-key`, $\mathcal{V}'$) is recorded, then record (`pub-key`, $\mathcal{V}$) and output (`pub-key`, $\mathcal{V}$) to $\mathcal{P}_0$.

Signing:

1. Upon receiving (`sign`, $sgid, m$) from $\mathcal{P}_i$:
   - If all signatories have provided a (`sign`, $sgid, m$) input, then record ($sgid, m$).
   - Send (`sign`, $sgid, m, i$) to $\mathcal{S}$.
2. Upon receiving (`sig-string`, $sgid, m, \sigma, \mathcal{X}$) from $\mathcal{S}$:
   - If ($sgid, m$) is not recorded, or ($sgid, m, \sigma, \ldots$) is already recorded, then end the activation.
   - Else, set $\beta = \mathcal{V}(m, \sigma)$ and record ($sgid, m, \sigma, \beta, \mathcal{C}$), where if $\beta = 0$ and $\mathcal{X}$ is the identity of a corrupted party, then $\mathcal{C} = \mathcal{X}$ (or $\mathcal{C}$ is the lexicographically first corrupted party if $\mathcal{X}$ is not corrupted); otherwise, if $\beta = 1$, then $\mathcal{C} = \emptyset$. '*Accountability*'

   Output (`sig-string`, $sgid, m, \sigma, \mathcal{X}$) to $\mathcal{P}_0$.

Verification:

Upon receiving (`sig-vrfy`, $m, \sigma, \mathcal{V}'$) from a party $\mathcal{Q}$, do:

- If $\mathcal{V}' = \mathcal{V}$, there is no ($sgid, m$) record for any $sgid$, the state is `uncompromised`, then output (`istrue`, $m, \sigma, 0$) to $\mathcal{Q}$. '*Unforgeability*'
- Else output (`istrue`, m, σ, β) to $\mathcal{Q}$, where $\beta = \mathcal{V}(m, \sigma)$. '*Consistency*'

Key-Refresh:

- Upon receiving input `key-refresh` from $\mathcal{P}_i \in \boldsymbol{P}$, send (`key-refresh`, i) to $\mathcal{S}$.
- Upon receiving `refresh-done` from $\mathcal{S}$, record all (`quarantined`) signatories as (`non-corrupted`) and increment c.
- Upon receiving a request for the value of the refresh counter c, return that value.

Corruption/Decorruption:

1. Upon receiving input (`corrupt`, $\mathcal{P}_j$) from $\mathcal{S}$, record $\mathcal{P}_j$ as (`corrupted`). If all signatories are recorded as either corrupted or quarantined then change state to `compromised`.
2. Upon receiving input (`decorrupt`, $\mathcal{P}_j$) from $\mathcal{S}$, record $\mathcal{P}_j$ as (`quarantined`).
3. Upon receiving input (`status`) from $\mathcal{P}_j$, return to $\mathcal{P}_j$ its corruption status.

Figure 15: Ideal Threshold Signature Functionality $\mathcal{F}_{\mathsf{tsig}}$

algorithm $\mathcal{V}$. Indeed, signatures cannot change status during the execution; if an invalid signature were to become valid, it would be equivalent to breaking unforgeability.

Beyond providing clear logic and simpler code for $\mathcal{F}_{\mathsf{tsig}}$, this formulation also represents the fact that in some cases the signature string generated by the system would be known the the corrupted parties (and thus to the environment) before they are known to the ideal-model adversary (i.e., to the simulator). Still, unforgeability guarantees that such situations happen only when all parties have agreed to sign the said message.

The administrative owner, $\mathcal{P}_0$. It is stressed that the administrative owner $\mathcal{P}_0$ has no 'cryptographic capabilities' and is unable to generate signatures or expose any sensitive information. Its roles are to (i) determine the identities of the parties and the *epid*; (ii) to aggregate the information provided by the parties to generate the public key and later the signatures.

The *sgid* mechanism. Recall that to generate a signature on a message m, all parties need to consistently provide $\mathcal{F}_{\mathsf{tsig}}$ with some identifier, *sgid*, associated with the message m. In contrast, the signature string returned to $\mathcal{P}_0$ does not include *sgid*, nor is a verification query required to provide the relevant *sgid*. The main reason for including this additional identifier is to help the parties distinguish between different signing requests that unintentionally (perhaps even unknowingly) mention the same message. Said otherwise, the *sgid* is a mechanism that puts on the user (or the owner) of $\mathcal{F}_{\mathsf{tsig}}$ the onus of distinguishing between different instances of the signature-generation process.

Modeling pre-signatures. It is noted that $\mathcal{F}_{\mathsf{tsig}}$ makes no mention of splitting the signature generation process to a pre-processing part and an online part. Indeed, while this capability is a main feature of our protocol, it mandates no mention in the definition of security. Said otherwise, the overall functionality of this mode is identical to that of standard threshold signatures. The difference is *only* in the various efficiency gains and costs.

Modeling strong unforgeability. The current formulation of $\mathcal{F}_{\mathsf{tsig}}$ allows for adversarial generation of new valid signature strings for a legitimately signed message. *Strong unforgeability,* namely disallowing the generation of new valid signature strings even for signed messages, can be captured by having $\mathcal{F}_{\mathsf{tsig}}$ count the number of times that a message m has been signed (presumably with a different *sgid* for each signature), and then limiting the number of different signature strings that can be accepted as legitimate signatures for that message.

Unforgeability vs availability. It is emphasized that, while $\mathcal{F}_{\mathsf{tsig}}$ provides robust unforgeability guarantees in the face of an adaptive, mobile adversary, it offers only limited guarantees regarding the availability, or liveness of the system. In particular, the only liveness guarantee is that signature verification is a local process that returns within a single activation. In contrast, the ideal adversary (i.e., the simulator) can arbitrarily delay the generation of the public key and of each signature. Furthermore, the simulator can arbitrarily delay the successful completion of proactive key refreshes. This formulation was chosen mainly for sake of simplicity, so as to allow for relatively simple protocols that allow highlighting the new algorithmic constructs. Formulating (and realizing) stronger variants of $\mathcal{F}_{\mathsf{tsig}}$ that provide better liveness guarantees is left out of scope for this work.

6.3 Security Claims

Theorem 6.1. *Assuming semantic security of the Paillier cryptosystem, strong-RSA assumption, DDH, and unforgeability (resp. enhanced unforgeability) of ECDSA, the protocol from Figure 4 (resp. Figure 5) UC-realizes functionality $\mathcal{F}_{\mathsf{tsig}}$ in the presence of the strict global random oracle functionality $\mathcal{H}$.*

The above theorem is a corollary of Lemma 6.5 and Theorems 6.6 and 6.7 stated further below. The proof follows by contraposition; we show that if our protocol does not UC-realize functionality $\mathcal{F}_{\mathsf{tsig}}$, then at least one of the following holds true: (i) there exists a PPTM that breaks strong RSA (ii) there exists a PPTM that breaks DDH (iii) there exists a PPTM that can distinguish Paillier ciphertexts (iv) there exists a PPT existential forger for the standard/enhanced, non-threshold ECDSA algorithm. Before that, we define *forgeries* and *fault misattributions*.

6.3.1 Forgeries & Fault Misattributions

Notation 6.2. Write Σ for the signing protocol from Figure 4 (resp. Figure 5). Let $\mathcal{Z}$ denote an arbitrary PPTM corrupting a subset of parties adaptively in an execution of Σ. It is assumed that $\mathcal{Z}$ does not corrupt all parties simultaneously in a given key-refresh epoch. We write $\tau^{\mathcal{H}}_{\mathcal{Z},\Sigma}$ to denote the *transcript* of the protocol in an arbitrary execution of Σ in the presence of $\mathcal{Z}$ and global random oracle $\mathcal{H}$, consisting of (i) the honest parties' inputs, (ii) the honest parties' outputs, i.e., signatures and/or corrupted parties' identities (iii) the messages exchanged between all parties and (iv) the output of $\mathcal{Z}$.

Definition 6.3 (Forgeries & Unforgeability)**.** Using Notation 6.2, we say that $\tau^{\mathcal{H}}_{\Sigma,\mathcal{Z}}$ contains a *forgery* if:

1. $\mathcal{Z}$ outputs (s, msg^*).
2. There is no record of $(\texttt{sign}, sgid, \text{msg}^*, \ldots)$ as a common input in $\tau^{\mathcal{H}}_{\Sigma,\mathcal{Z}}$ for some $sgid$.
3. $\texttt{vrfy}_{\tau^{\mathcal{H}}_{\Sigma,\mathcal{Z}}}(s, \text{msg}^*) = 1$, where $\texttt{vrfy}_{\tau^{\mathcal{H}}_{\Sigma,\mathcal{Z}}}(\cdot)$ denotes the signature verification process associated with $\tau^{\mathcal{H}}_{\Sigma,\mathcal{Z}}$.

Furthermore, we say that Σ is *unforgeable* if the the probability that $\tau^{\mathcal{H}}_{\Sigma,\mathcal{Z}}$ contains a forgery, resulting from an arbitrary execution of Σ in the presence of $\mathcal{Z}$, is negligible for every $\mathcal{Z}$.

Definition 6.4 (Fault Misattributions & Accountability)**.** Using the notation from Notation 6.2, we say that $\tau^{\mathcal{H}}_{\Sigma,\mathcal{Z}}$ contains a *fault misattribution* if there exists some $sgid$ in $\tau^{\mathcal{H}}_{\Sigma,\mathcal{Z}}$ such that two honest parties, $\mathcal{P}_i$ and $\mathcal{P}_j$, output $(\texttt{sig-string}, sgid, m, \sigma, \mathcal{X})$ and $(\texttt{sig-string}, sgid, m, \sigma', \mathcal{X}')$ respectively, with $(\sigma, \mathcal{X}) \neq (\sigma', \mathcal{X}')$. (Informally, there exists an aborted signature session in $\tau^{\mathcal{H}}_{\Sigma,\mathcal{Z}}$ where the honest parties fail to reach consensus on the identity of at least one corrupted party.) Furthermore, we say that Σ is *accountable* if the probability that $\tau^{\mathcal{H}}_{\Sigma,\mathcal{Z}}$ contains a fault misattribution, resulting from an arbitrary execution of Σ in the presence of $\mathcal{Z}$, is negligible for every $\mathcal{Z}$.

6.3.2 Putting Everything Together

Lemma 6.5. *Using the notation above, if Σ is unforgeable and accountable according to Definitions 6.3 and 6.4, then Σ UC-realizes functionality $\mathcal{F}_{\mathsf{tsig}}$ in the presence of the global random oracle functionality $\mathcal{H}$.*

Proof. Assume that Σ is unforgeable and accountable according to Definitions 6.3 and 6.4, and consider the following ideal-world simulator interacting with $\mathcal{F}_{\mathsf{tsig}}$, $\mathcal{Z}$, and global random oracle $\mathcal{H}$. Following the standard paradigm, and without loss of generality, any action performed by the corrupted parties is assumed to be performed by $\mathcal{Z}$. Furthermore, it is assumed that $\mathcal{S}$ emulates the ideal synchronous communication functionality $\mathcal{F}_{\text{syn}}$, which in the real world is used for the communication of all messages (see Section 3.5). Since the simulator does not perform any special operations with respect to $\mathcal{F}_{\text{syn}}$, it can be considered as a separate machine with the exact same behaviour as in the real world.

In a nutshell, $\mathcal{S}$ simply runs the code of the honest parties when interacting with $\mathcal{Z}$, and interacts with $\mathcal{H}$ only as prescribed by the simulated honest parties. More specifically, $\mathcal{S}$ performs the following actions when interacting with $\mathcal{Z}$ and $\mathcal{F}_{\mathsf{tsig}}$:

1. When obtaining $(\texttt{keygen}, i)$ for an uncorrupted party $\mathcal{P}_i$ from $\mathcal{F}_{\mathsf{tsig}}$, $\mathcal{S}$ starts the key generation for $\mathcal{P}_i$.
2. When obtaining $(\texttt{pub-key}, sid)$ from $\mathcal{F}_{\mathsf{tsig}}$, return $(\texttt{pub-key}, \mathcal{V}_X)$ where X is the ECDSA public key resulting from the interaction between $\mathcal{S}$ and $\mathcal{Z}$, and $\mathcal{V}_X$ is the verification algorithm associated with X.
3. When obtaining $(\texttt{sign}, sgid, m, i)$ for an uncorrupted party $\mathcal{P}_i$ from $\mathcal{F}_{\mathsf{tsig}}$, $\mathcal{S}$ initiates the signing phase for $\mathcal{P}_i$. It is important to note that, depending on the instructions from $\mathcal{Z}$, the signing phase may involve either the fully online interactive protocol or the non-interactive protocol, which uses presignatures generated in a previous step between $\mathcal{S}$ and $\mathcal{Z}$.
4. Once $\mathcal{S}$ receives $(\texttt{sign}, sgid, m, i)$ for all $\mathcal{P}_i$, the simulator sends $(\texttt{sig-string}, sgid, m, \sigma, \mathcal{X})$ to $\mathcal{F}_{\mathsf{tsig}}$ where $\mathcal{X} = \emptyset$ if the interaction between $\mathcal{S}$ and $\mathcal{Z}$ resulted in a valid signature σ for m. If the signature session was aborted by $\mathcal{Z}$, then σ and $\mathcal{X}$ represent the failed verification and the identity of the exposed party, as determined by the simulated honest parties.
5. When obtaining $(\texttt{key-refresh}, i)$ from $\mathcal{F}_{\mathsf{tsig}}$, $\mathcal{S}$ starts the key refresh for $\mathcal{P}_i$.

6. Once the key-refresh between $\mathcal{S}$ and $\mathcal{Z}$ concludes successfully, $\mathcal{S}$ sends `refresh-done` to $\mathcal{F}_{\mathsf{tsig}}$.

   Handling corruptions. If $\mathcal{Z}$ decides to corrupt a party, say $\mathcal{P}_i$, $\mathcal{S}$ sends $(\texttt{corrupt}, \mathcal{P}_i)$ to $\mathcal{F}_{\mathsf{tsig}}$ and reveals the secret state (ECDSA secret share, Paillier private key) to $\mathcal{Z}$. When $\mathcal{Z}$ decides to decorrupt a previously corrupted $\mathcal{P}_i$, $\mathcal{Z}$ restores the state of $\mathcal{P}_i$, and $\mathcal{S}$ sends $(\texttt{decorrupt}, \mathcal{P}_i)$ to $\mathcal{F}_{\mathsf{tsig}}$. This allows $\mathcal{S}$ to continue the interaction with $\mathcal{Z}$ without issue by simply running the code of the honest $\mathcal{P}_i$ from that point onward.

It is easy to see that the ideal experiment above is trivial in the sense that $\mathcal{Z}$'s view of Σ in the real and ideal experiments is identically distributed. The only way the two experiments may diverge is with respect to the ideal functionality, namely in the following two ways:

(a) There was an aborted simulated signature session for some *sgid* and m, and $\mathcal{S}$ failed to register a valid tuple $(\texttt{sig-string}, \mathit{sgid}, m, \sigma, \mathcal{X})$ because there was no consensus on $\mathcal{X}$ by the simulated honest parties.

(b) $\mathcal{Z}$ produced a pair $(\mathrm{msg}^*, \sigma^*)$ such that $\mathcal{V}_X(\mathrm{msg}^*, \sigma^*) = 1$ but $\mathcal{F}_{\mathsf{tsig}}$ returns $(\texttt{istrue}, \mathrm{msg}^*, \sigma^*, 0)$ on input $(\texttt{sig-vrfy}, \mathrm{msg}^*, \sigma^*, \mathcal{V}_X)$.

Notice that Item (a) contradicts the presumed accountability of Σ, and Item (b) contradicts the presumed unforgeability of Σ. This concludes the proof. □

Theorem 6.6. *Under DCR, DDH, strong RSA and unforgeability (resp. enhanced unforgeability) of ECDSA, it holds that protocol Σ from Figure 4 (resp. Figure 5) is unforgeable as per Definition 6.3.*

The proof of Theorem 6.6 is presented in Section 7.

Theorem 6.7. *Under the strong RSA assumption, it holds that protocol Σ from Figure 4 (resp. Figure 5) is accountable as per Definition 6.4.*

The proof of Theorem 6.7 is very similar to the proof of Theorem 6.6; it is included in Appendix B.

7 Proof of Theorem 6.6

In this section, we prove Theorem E.1 by carefully defining several intermediate hybrid experiments that reduce the unforgeability of the protocol to the underlying cryptographic assumptions: Strong RSA, DDH, DCR, and ECDSA security. To maintain the structure outlined in the introduction (Section 2.6), the analysis is divided into three subsections: the first subsection (the A-experiments) contains the reduction to Strong RSA, the second (the B-experiments) covers the reductions to DDH and DCR, and the final one (the C-experiments) addresses the reduction to ECDSA security, which concludes the proof.

All of the security hops in this section are fairly standard, except for the first experiment A_0, which might appear somewhat mysterious. To address this, we have added clarifications in the relevant subsections. Before beginning the analysis, we make a few simplifying assumptions and provide justifications for them.

Simplifying assumptions As noted in Remark 4.1, within a specific session identified by the *sid*, the key-refresh phase is assumed to be executed sequentially, meaning that no other protocol components for this *sid*, including another concurrent execution of the key-refresh, are executed while the key-refresh is ongoing. However, no other limitations are imposed on the concurrency of other components. Additionally, we make the following simplifying assumptions about $\mathcal{Z}$:

1. For simplicity, we assume that $\mathcal{Z}$ statically corrupts all but one party, and the honest party is denoted $\mathcal{P}_b$. Later, in Section 8, we provide a formal reduction from the adaptive to the static corruption model.

2. $\mathcal{Z}$ *does not* check whether $\mathcal{P}_b$'s messages are consistent with the random oracle, i.e. it does not verify that $\mathcal{P}_b$'s hashes are computed correctly. This assumption does not incur any loss of generality and is only used to avoid keeping track of real versus programmed oracle queries in the security reduction.

3. $\mathcal{Z}$ may only *abort* (i.e. quit the execution of the protocol all together) during the third round of the key-refresh protocol. This assumption does not incur any loss of generality, as it can be handled by defining an arbitrary continuation of the protocol up to the third round of the key-refresh epoch.

4. When $\mathcal{Z}$ discontinues the execution of a key-refresh phase, it is assumed that $\mathcal{Z}$ aborts at this point (i.e. quits the execution of the protocol altogether). This assumption results in a multiplicative inverse-polynomial loss in the forging probability (specifically by the number of key-refresh epochs), which we justify via Lemma 7.26 in Section 7.3.1.

5. $\mathcal{Z}$ never sends a failing NIZK or any other publicly verifiable failing message, but rather sends a special discontinuation symbol (denoted $\perp$). It's important to note that the discontinuation of a given signing session does not necessarily imply an abort, as aborting refers to quitting the protocol altogether (discontinuation of the key-refresh does imply an abort, as per Item 4 above and Remark 4.2). This assumption does not incur any loss of generality, as the NIZK/message in question can be verified by the attacker before it is sent.

Finally, it is assumed that $\mathcal{Z}$ finds no collisions in the random oracle and that the same *epid* is never sampled again in different key-refresh epochs. This last assumption incurs a small, negligible penalty that we ignore in the analysis below.

Running time of the hybrid experiments. The running time of each algorithm in the experiments presented below is essentially identical to that of the honest party. This fact is evident from the description of the algorithms. Finally, the security parameters and all dependent variables are assumed to be fixed.

7.1 Reduction to Strong-RSA

7.1.1 Hybrid A_0

FIGURE 16 (Hybrid A_0, i.e. Forging Experiment A_0)

Define $\mathcal{R}_{\mathrm{A}_0}$ interacting with $\mathcal{Z}$ corrupting all players but $\mathcal{P}_b$:

$\mathcal{R}_{\mathrm{A}_0}$ runs the code of $\mathcal{P}_b$ with the following differences.

1. *Key-Generation & Aux Info.*
   (a) Fork the protocol by using fresh randomness for $\mathcal{P}_b$'s second and third-round messages.
   This is accomplished by faking $\mathcal{P}_b$'s decommitment in the second round. (For the security analysis, it is assumed that the decommitment is faked, meaning it is inconsistent with the RO in both the main and forked paths. All other queries are consistent with the RO.)
   (b) Discard one of the executions according to the following criterion:
      i. If the second, 'forked' path aborts, continue on the second path.
      ii. Else, continue on the first, 'main' path.
2. *Presigning & Signing.*
   Same as as the real experiment.

Output. Output the transcript of the execution.

Figure 16: Hybrid A_0, i.e. Forging Experiment A_0

Remark 7.1 (The purpose of experiment A_0). Experiment A_0 may seem a bit odd, as the experiment is biased towards an abort. Notice that $\mathcal{R}_{\mathrm{A}_0}$ never continues on the forked path unless it aborts, in which case $\mathcal{R}_{\mathrm{A}_0}$ is instructed to choose the forked path if the main path does not abort as well. The purpose of this experiment is related to the extraction of the adversary's secrets in later experiments. Specifically, both paths need to be non-aborting for the later experiments to continue. For this reason, we instruct $\mathcal{R}_{\mathrm{A}_0}$ to choose the forked path if it aborts and the main one does not and terminate the experiment at this stage.

Proposition 7.2. *Suppose that $\mathcal{Z}$ outputs a forgery with probability α in an execution of Σ, then $\mathcal{Z}$ outputs a forgery with probability at least α^2 in Experiment A_0 from Figure 16.*

By Jensen's inequality, the above is an immediate corollary of the proposition below.

Proposition 7.3. *Assume that all random oracle answers (i.e., all answers for all possible queries) are sampled in advance and fixed, and let $\mathcal{Z}$'s random tape be set to some arbitrary string $\rho \in \{0,1\}^*$. Denote the corresponding machine by $\mathcal{Z}(\rho)$. If $\mathcal{Z}(\rho)$ outputs a forgery with probability α in an execution of Σ, then $\mathcal{Z}(\rho)$ outputs a forgery with probability at least α^2 in Experiment A_0 from Figure 16.*

7.1.2 Proof of Proposition 7.3

To alleviate notation, we ignore the random tape of $\mathcal{Z}$ which is assumed to be fixed to $\rho \in \{0,1\}^*$.

Definition 7.4. Define protocol Σ^* similarly to Σ, except that early aborts by $\mathcal{Z}$ in Σ resulting in forgeries are not labeled as 'aborts'. Specifically, if $\mathcal{Z}$ halts prematurely in Σ^* and outputs a forgery, the protocol continues arbitrarily (e.g., with parties exchanging a check mark symbol '✓') until the end of the final key-refresh epoch. Furthermore, if $\mathcal{Z}$ does not halt prematurely and the execution does not result in a forgery, a final message is added to the protocol at the end of the final key-refresh epoch, and it is assumed that $\mathcal{Z}$ aborts Σ^* in this final message. Finally, define experiment A_0^* with respect to Σ^* as the analogue of A_0 with respect to Σ.

Remark 7.5. We point out that A_0 and A_0^* are not strictly equivalent. For instance, A_0 may yield a forgery (because $\mathcal{Z}$ aborts prematurely in the forked path and outputs a forgery), while A_0^* does not (because the experiment continued on the main path and did not yield a forgery).

Claim 7.6. *Let $\boldsymbol{\tau}_\Sigma$ and $\boldsymbol{\tau}_{\mathrm{A}_0}$ denote the transcripts of Σ and A_0, respectively, viewed as random variables over the honest party's messages and the RO answers, and define $\boldsymbol{\tau}_{\Sigma^*}$ and $\boldsymbol{\tau}_{\mathrm{A}_0^*}$ analogously. Write $\mathsf{forge}_{\mathrm{X}}$ for $\mathrm{X} \in \{\Sigma, \Sigma^*, \mathrm{A}_0, \mathrm{A}_0^*\}$ to denote the forgery event in the relevant experiment, and let $\alpha = \Pr_{\boldsymbol{\tau}_\Sigma}[\mathsf{forge}_\Sigma]$. Then:*

1. $\Pr_{\boldsymbol{\tau}_{\Sigma^*}}[\mathsf{forge}_{\Sigma^*}] = \alpha$
2. $\Pr_{\boldsymbol{\tau}_{\mathrm{A}_0}}[\mathsf{forge}_{\mathrm{A}_0}] \geq \Pr_{\boldsymbol{\tau}_{\mathrm{A}_0^*}}[\mathsf{forge}_{\mathrm{A}_0^*}]$

Proof. Item 1 is immediate. For Item 2, it suffices to note that A_0 and A_0^* differ only when there is a premature abort *together with a forgery*, i.e. executions of A_0 and A_0^* that do not contain such an event are indistinguishable. Therefore, any execution of A_0 that yields a forgery also yields a forgery according to the rules of A_0^*. □

Next, we introduce additional notation and we state and prove some intermediate claims. Let $\boldsymbol{\tau}_{\Sigma^*,1}, \ldots, \boldsymbol{\tau}_{\Sigma^*,m}$ denote the partial transcripts of Σ^*, viewed as random variables, and let '$\succ$' denote a partial ordering of the transcripts such that $u_i \succ u_{i-1}$ if $(u_i, u_{i-1}) \in \mathrm{supp}(\boldsymbol{\tau}_{\Sigma^*,i}) \times \mathrm{supp}(\boldsymbol{\tau}_{\Sigma^*,i-1})$ and u_i is a valid continuation of u_{i-1}. Next, define the sequences of events $\{\mathsf{go}_{\Sigma^*,i}, \mathsf{abort}_{\Sigma^*,i}\}_{i=1}^m$ where:

1. $\mathsf{go}_{\Sigma^*,i}$ denotes the event that the attacker has not aborted Σ^* up-to-and-including the i-th iteration.
2. $\mathsf{abort}_{\Sigma^*,i}$ denotes that the attacker aborts at the i-th iteration of the experiment.

Finally, define random variable $\boldsymbol{\varepsilon}_{\Sigma^*,i}$ such that $\boldsymbol{\varepsilon}_{\Sigma^*,i} \leftarrow \Pr[\mathsf{go}_{\Sigma^*,i} \mid \boldsymbol{\tau}_{\Sigma^*,i} = u_i]$ for $u_i \leftarrow \boldsymbol{\tau}_{\Sigma^*,i}$ and let f s.t.

$$f(u_i, u_{i-1}) = \begin{cases} \Pr[\boldsymbol{\tau}_{\Sigma^*,i} = u_i \mid \boldsymbol{\tau}_{\Sigma^*,i-1} = u_{i-1} \wedge \mathsf{go}_{\Sigma^*,i-1}] & \text{if } \boldsymbol{\varepsilon}_{\Sigma^*,i-1}(u_{i-1}) \neq 0 \\ \Pr[\boldsymbol{\tau}_{\Sigma^*,i} = u_i \mid \boldsymbol{\tau}_{\Sigma^*,i-1} = u_{i-1}] & \text{otherwise.} \end{cases}$$

Claim 7.7. *It holds that $\sum_{u_m \succ \ldots \succ u_1} f(u_1) \cdot f(u_2, u_1) \ldots f(u_m, u_{m-1}) = 1$*

Claim 7.8. *It holds that $\alpha = \mathop{\mathbf{E}}_{\boldsymbol{f}}[\boldsymbol{\varepsilon}_1 \ldots \boldsymbol{\varepsilon}_m]$ for $\boldsymbol{f}$ such that for arbitrary $u_{\Sigma^*} = u_m \succ \ldots \succ u_m$:*

$$\Pr[\boldsymbol{f} = u_{\Sigma^*}] = f(u_1) \cdot f(u_2, u_1) \ldots f(u_m, u_{m-1}).$$

Claim 7.9. *It holds that $\Pr_{\boldsymbol{\tau}_{\mathrm{A}_0^*}}[\mathsf{forge}_{\mathrm{A}_0^*}] = \mathbf{E}_{\boldsymbol{f}}[\boldsymbol{\varepsilon}^2_{\Sigma^*,1} \ldots \boldsymbol{\varepsilon}^2_{\Sigma^*,m}]$.*

Proofs of Claims 7.7 to 7.9. See Section 7.1.3. □

Putting everything together. Proposition 7.3 is a straightforward corollary of the above claims. Namely, by Claim 7.6 and Claim 7.9, we know that $\Pr_{\boldsymbol{\tau}_{\mathrm{A}_0}}[\mathsf{forge}_{\mathrm{A}_0}] \geq \mathbf{E}_{\boldsymbol{f}}[\boldsymbol{\varepsilon}^2_{\Sigma^*,1} \dots \boldsymbol{\varepsilon}^2_{\Sigma^*,m}]$, and, by Jensen's inequality, Claim 7.8 and Claim 7.6, we deduce that:

$$\begin{aligned}\Pr_{\boldsymbol{\tau}_{\mathrm{A}_0}}[\mathsf{forge}_{\mathrm{A}_0}] &\geq \mathbf{E}_{\boldsymbol{f}}[\boldsymbol{\varepsilon}^2_{\Sigma^*,1} \dots \boldsymbol{\varepsilon}^2_{\Sigma^*,m}] \\ &\geq \mathbf{E}_{\boldsymbol{f}}[\boldsymbol{\varepsilon}_{\Sigma^*,1} \dots \boldsymbol{\varepsilon}_{\Sigma^*,m}]^2 \\ &= \Pr_{\boldsymbol{\tau}_{\Sigma^*}}[\mathsf{forge}_{\Sigma^*}]^2 = \alpha^2\end{aligned}$$

This concludes the proof of Proposition 7.3. □

7.1.3 Proofs of Claims 7.7 to 7.9

Proof of Claim 7.7. Fix $u_{i-1} \succ \dots \succ u_1$. Deduce that

$$\begin{aligned}&\sum_{u_i \succ u_{i-1}} f(u_1) \cdot f(u_2, u_1) \dots f(u_{i-1}, u_{i-2}) \cdot f(u_i, u_{i-1}) \\ &\qquad = f(u_1) \cdot f(u_2, u_1) \dots f(u_{i-1}, u_{i-2}) \sum_{u_i \succ u_{i-1}} f(u_i, u_{i-1}) \\ &\qquad = f(u_1) \cdot f(u_2, u_1) \dots f(u_{i-1}, u_{i-2}).\end{aligned}$$

The claim follows by simple descent since $\sum_{u_1} f(u_1) = 1$. □

Proof of Claim 7.8. Define auxiliary functions $g(u_i, u_{i-1})$ and $h(u_{i-1})$ such that

$$g(u_i, u_{i-1}) = \begin{cases} \Pr[\boldsymbol{\tau}_{\Sigma^*,i} = u_i \wedge \bigwedge_{j=i}^m \mathsf{go}_j \mid \boldsymbol{\tau}_{\Sigma^*,i-1} = u_{i-1} \wedge \mathsf{go}_{\Sigma^*,i-1}] & \text{if } \boldsymbol{\varepsilon}_{\Sigma^*,i-1}(u_{i-1}) \neq 0 \\ 0 \quad \text{otherwise.} \end{cases}$$

$$h(u_{i-1}) = \begin{cases} \Pr[\bigwedge_{j=i}^m \mathsf{go}_j \mid \boldsymbol{\tau}_{\Sigma^*,i-1} = u_{i-1} \wedge \mathsf{go}_{\Sigma^*,i-1}] & \text{if } \boldsymbol{\varepsilon}_{\Sigma^*,i-1}(u_{i-1}) \neq 0 \\ 0 \quad \text{otherwise.} \end{cases}$$

Observe that for $u_i \succ u_{i-1}$:

$$\begin{cases} \Pr[\bigwedge_{j=i}^m \mathsf{go}_{\Sigma^*,i} \mid \boldsymbol{\tau}_{\Sigma^*,i} = u_i] = \boldsymbol{\varepsilon}_{\Sigma^*,i}(u_i) \cdot h(u_i) \\ h(u_i) = \sum_{u_{i+1} \succ u_i} g(u_{i+1}, u_i) \\ g(u_i, u_{i-1}) = \Pr[\bigwedge_{j=i}^m \mathsf{go}_{\Sigma^*,i} \mid \boldsymbol{\tau}_{\Sigma^*,i} = u_i] \cdot f(u_i, u_{i-1}) \end{cases}$$

Thus:

$$\begin{aligned}\alpha = \Pr[\bigwedge_{i=1}^m \mathsf{go}_{\Sigma^*,i}] &= \sum_{u_1} \Pr[\bigwedge_{i=1}^m \mathsf{go}_{\Sigma^*,i} \mid \boldsymbol{\tau}_{\Sigma^*,1} = u_1] \cdot \Pr[\boldsymbol{\tau}_{\Sigma^*,1} = u_1] \\ &= \sum_{u_1} h(u_1) \cdot \boldsymbol{\varepsilon}_{\Sigma^*,1}(u_1) \cdot f(u_1) \\ &= \sum_{u_1} \boldsymbol{\varepsilon}_{\Sigma_1^*}(u_1) \cdot f(u_1) \sum_{u_2 \succ u_1} g(u_2, u_1) \\ &= \sum_{u_1} \boldsymbol{\varepsilon}_{\Sigma^*,1}(u_1) \cdot f(u_1) \sum_{u_2 \succ u_1} \Pr[\bigwedge_{i=2}^m \mathsf{go}_{\Sigma^*,i} \mid \boldsymbol{\tau}_{\Sigma^*,2} = u_2] \cdot f(u_2, u_1) \\ &= \sum_{u_m \succ \dots \succ u_1} \boldsymbol{\varepsilon}_{\Sigma^*,1}(u_1) \dots \boldsymbol{\varepsilon}_{\Sigma^*,m}(u_m) \cdot f(u_1) \cdot f(u_2, u_1) \dots f(u_m, u_{m-1}) \\ &= \mathop{\mathbf{E}}_{\boldsymbol{f}}[\boldsymbol{\varepsilon}_1 \dots \boldsymbol{\varepsilon}_m]\end{aligned}$$

□

Proof of Claim 7.9. Define $\{\boldsymbol{\tau}_{\mathrm{A}_0^*,i}, \mathsf{go}_{\mathrm{A}_0^*,i}\}_{i=1}^m$ analogously to $\{\boldsymbol{\tau}_{\Sigma^*,i}, \mathsf{go}_{\Sigma^*,i}\}_{i=1}^m$ with respect to Experiment A_0^*. Notice that if for all i and $u_i \in \mathrm{supp}(\boldsymbol{\tau}_{\Sigma^*,i})$, it holds that $\Pr[\mathsf{go}_{\mathrm{A}_0^*,i} \mid \boldsymbol{\tau}_{\mathrm{A}_0^*,i} = u_i] = \boldsymbol{\varepsilon}_{\Sigma^*,i}(u_i)^2$, then, using the same calculation as in Claim 7.8, it follows that $\Pr_{\boldsymbol{\tau}_{\mathrm{A}_0^*}}[\mathsf{forge}_{\mathrm{A}_0^*}] = \mathbf{E}_{\boldsymbol{f}}[\boldsymbol{\varepsilon}^2_{\Sigma^*,1} \dots \boldsymbol{\varepsilon}^2_{\Sigma^*,m}]$. So, the main challenge is showing that $\Pr[\mathsf{go}_{\mathrm{A}_0^*,i} \mid \boldsymbol{\tau}_{\mathrm{A}_0^*,i} = u_i] = \boldsymbol{\varepsilon}_{\Sigma^*,i}(u_i)^2$, and we dedicate the rest of the proof to this. Write $\mathsf{go}_{\mathrm{M},i}$ and $\mathsf{go}_{\mathrm{F},i}$ for the event that $\mathcal{Z}$ aborts in the main and forked path of experiment A_0^*. Clearly, $\boldsymbol{\varepsilon}_{\Sigma^*,i}(u_i) = \Pr[\mathsf{go}_{\mathrm{M},i} \mid \boldsymbol{\tau}_{\mathrm{A}_0^*,i} = u_i] = \Pr[\mathsf{go}_{\mathrm{F},i} \mid \boldsymbol{\tau}_{\mathrm{A}_0^*,i} = u_i]$ and

$$\Pr[\mathsf{go}_{\mathrm{A}_0^*,i} \mid \boldsymbol{\tau}_{\mathrm{A}_0^*,i} = u_i] = \Pr[\mathsf{go}_{\mathrm{M},i} \wedge \mathsf{go}_{\mathrm{F},i} \mid \boldsymbol{\tau}_{\mathrm{A}_0^*,i} = u_i] \tag{1}$$

Given that $\mathcal{Z}$'s random tape and all random oracle queries are fixed, it follows that $\mathsf{go}_{\mathrm{F},i}$ and $\mathsf{go}_{\mathrm{M},i}$ are determined by $\mathcal{P}_b$'s second and third-round messages in their respective paths. Consequently, it holds that, conditioned on $\boldsymbol{\tau}_{\mathrm{A}_0^*,i} = u_i$, the events $\mathsf{go}_{\mathrm{M},i}$ and $\mathsf{go}_{\mathrm{F},i}$ are independent, as $\mathcal{P}_b$'s messages are independent after the fork. (Using the fact that for any independent identically distributed random variables $\boldsymbol{a}$ and $\boldsymbol{a}'$, and any boolean function b, $\Pr[b(\boldsymbol{a}) = 1 \wedge b(\boldsymbol{a}') = 1] = \Pr[b(\boldsymbol{a}) = 1] \cdot \Pr[b(\boldsymbol{a}') = 1]$.) This concludes the proof. □

7.1.4 Hybrid A_1

FIGURE 17 (Hybrid A_1, i.e. Forging Experiment A_1)

Define $\mathcal{R}_{\mathrm{A}_1}$ interacting with $\mathcal{Z}$ corrupting all players but $\mathcal{P}_b$ as follows:

$\mathcal{R}_{\mathrm{A}_1}$ runs the code of $\mathcal{R}_{\mathrm{A}_0}$ with the following difference:

1. All of $\mathcal{P}_b$'s ZK-proofs are generated by invoking the relevant ZK simulator.
2. (a) Every time the experiment is forked, do:
   i. If $\mathcal{Z}$ does not abort in the main path, then:
   – In the forked path, program the RO queries of the form $(\dots, N_j)$ as per Section 5.2.1.
   – Extract the Paillier keys using the relevant extractor.
   ii. Else, ignore Item 2a.
   (b) If the experiment continues beyond the i-th key-refresh:
   Extract the i-th ECDSA key-shares using the special-soundness extractors.

Output. If $\mathcal{R}_{\mathrm{A}_1}$ fails to extract the i-th key-shares or the Paillier keys, then output fail.

Else output the transcript of the execution.

Figure 17: Hybrid A_1, i.e. Forging Experiment A_1

Proposition 7.10. *Let ν and ν' denote smallest (statistical) HVZK and soundness parameters in the protocol and let d and d' denote upper bounds on, respectively, the total number of ZK proofs and the number of RO queries. Suppose that $\mathcal{Z}$ outputs a forgery with probability α in Experiment A_0 from Figure 16, then $\mathcal{Z}$ outputs a forgery with probability $\alpha - \nu \cdot d - \nu' \cdot d'$ in Experiment A_1 from Figure 17.*

7.1.5 Proof of Proposition 7.10

Define in-between experiment A_0^* relative to Item 1 of Figure 17 as follows: $\mathcal{R}_{\mathrm{A}_0^*}$ proceeds as $\mathcal{R}_{\mathrm{A}_0}$ except that all of $\mathcal{P}_b$'s ZK-proofs are generated by invoking the relevant ZK simulator. Let $\alpha_0, \alpha_0^*, \alpha_1$ denote the probability that $\mathcal{Z}$ outputs a forgery in experiments A_0, A_0^*, A_1, and let $\mathsf{forge}_{\mathrm{X}}$ denote the forgery event in experiment $\mathrm{X} \in \{\mathrm{A}_0, \mathrm{A}_0^*, \mathrm{A}_1\}$. Proposition 7.10 follows from the claim below.

Claim 7.11. *Using the notation above, it holds that*

1. $\alpha_0^* \geq \alpha_0 - \nu \cdot d$
2. $\alpha_1 \geq \alpha_0^* - \nu' \cdot d'$.

Proof. For the first item, observe that A_0 and A_0^* are distinguishable only if the completeness property does not hold for one of the real ZK proofs. This happens with probability at most $\nu \cdot d$ which conclude the proof of Item 1. For the second item, A_0^* and A_1 are distinguishable in the following cases.

- $(N_j, \hat{N}_j, s_j, t_j)$ is malformed and the main path of the experiment does not abort.
- $(N_j, \hat{N}_j, s_j, t_j)$ is well-formed, the main path does not abort, but $\mathcal{R}_{\mathrm{A}_1}$ fails to extract the Paillier keys.

As either event occurs with probability at most $d' \cdot \nu'$, this concludes the proof for Item 2. □

7.1.6 Hybrid A_2

FIGURE 18 (Hybrid A_2, i.e. Forging Experiment A_2)

Define $\mathcal{R}_{\mathrm{A}_2}$ interacting with $\mathcal{Z}$ corrupting all players but $\mathcal{P}_b$.

Sample $(x, X) \leftarrow \mathbb{F}_q \times \mathbb{G}$ such that $X = g^x$. Proceed as $\mathcal{R}_{\mathrm{A}_1}$ except that:

1. *Key-Generation & Aux Info.*
   (a) Extract $\{X_j\}_{j \neq b}$ from the RO queries and set $X_b = X \cdot (\prod_{j \neq b} X_j)^{-1}$.
   (b) Extract $\{x_j\}_{j \neq b}$ and $\{\varphi(N_j)\}_{j \neq b}$ as in experiment A_1 in Figure 17.
   (c) Set all other items as in Figure 17.
2. *Presigning.*
   (a) Extract $\{\alpha_{j,b}, \hat{\alpha}_{j,b}, k_j, \gamma_j, \beta_{j,b}, \hat{\beta}_{j,b}\}_{j \neq b}$ by decrypting $\{D_{j,b}, \hat{D}_{j,b}, K_j, G_j, F_{b,j}, \hat{F}_{b,j}\}_{j \neq b}$.
   (b) Set $\delta_b = k_b \cdot \gamma_b + \gamma_b \cdot (\sum_{j \neq b} k_j) + k_b \cdot (\sum_{j \neq b} \gamma_j) - \sum_{j \neq b} \alpha_{j,b} + \beta_{j,b} \mod q$
   (c) Set all other items as in Figure 17.
3. *Signing.*
   (a) $\sigma_b = \left(k_b \cdot m + r \cdot \left(k_b \cdot x_b + x_b \cdot (\sum_{j \neq b} k_j) + k_b \cdot (\sum_{j \neq b} x_j) - \sum_{j \neq b} \hat{\alpha}_{j,b} + \hat{\beta}_{j,b}\right)\right) / \delta \mod q$.
   (b) Set all other items as in Figure 17.

Output. Same as $\mathcal{R}_{\mathrm{A}_1}$ in Figure 17.

Figure 18: Hybrid A_2, i.e. Forging Experiment A_2

Proposition 7.12. *Let α denote the probability that $\mathcal{Z}$ outputs a forgery in Experiment A_1 from Figure 18 and assume that $\mathcal{Z}$ outputs a forgery with probability less than $\alpha - \delta$ in Experiment A_2 from Figure 18. Let d and ℓ denote a bound on the number of RO queries and the bit length of the output space of the RO. Let ν denote the statistical soundness parameter of Π^{elog}. Then, there exists efficient PPTM $\mathcal{A}$ such that:*

$$\Pr_{(N,C) \leftarrow \mathsf{sRSA}(1^\kappa)} [(m, e) \leftarrow \mathcal{A}(1^\kappa, N, C) \ \ s.t. \ \ m^e = C \mod N \wedge e \notin \{1, -1\}] \geq (\delta - 3d\nu)^2 / 9d - 2^{-\ell}$$

7.1.7 Proof of Proposition 7.12

Define the intermediate experiment A_1^*, where $\mathcal{R}_{\mathrm{A}_1^*}$ proceeds as $\mathcal{R}_{\mathrm{A}_1}$ from Figure 17, except that it (inefficiently) checks if $(\Gamma_j, g, B_{j,1}, B_{j,2}, Y_j) \in \boldsymbol{R}_{\mathsf{elog}}$ for $j \neq b$. If a malformed tuple is detected, it halts the experiment and outputs an error. Define A_2^* and $\mathcal{R}_{\mathrm{A}_2^*}$ analogously with respect to A_2 and $\mathcal{R}_{\mathrm{A}_2}$ from Figure 18.

Let $\mathsf{forge}_{\mathrm{X}}$ and $\boldsymbol{\tau}_{\mathrm{X}}$ denote the forgery event and the transcript of $\mathrm{X} \in \{\mathrm{A}_1, \mathrm{A}_1^*, \mathrm{A}_2, \mathrm{A}_2^*\}$. Recall the relations $\boldsymbol{R}_{\mathsf{fac}}, \boldsymbol{R}_{\mathsf{enc\text{-}elg}}, \boldsymbol{R}_{\mathsf{aff\text{-}g}}$ from Section 3.2, i.e. the 'range-proof' relations, verifying the validity of N_j and the tuples $(\boldsymbol{I}_\varepsilon, K_j, Y_j, A_{j,1}, A_{j,2})$, $(\boldsymbol{I}_\varepsilon, G_j, Y_j, B_{j,1}, B_{j,2})$ and $(\boldsymbol{I}_\varepsilon, \boldsymbol{J}_\varepsilon, D_{j,i}, K_j, F_{j,i}, \Gamma_i)$, $(\boldsymbol{I}_\varepsilon, \boldsymbol{J}_\varepsilon, \hat{D}_{j,i}, K_j, \hat{F}_{j,i}, X_i)$ respectively.

Claim 7.13. *Experiments A_1 and A_1^*, and experiments A_2 and A_2^*, are statistically $(d \cdot \nu)$-indistinguishable.*

Proof. Corollary of the statistical soundness of Π^{elog}. □

Claim 7.14. *For* $\mathrm{X} \in \{\mathrm{A}_1^*, \mathrm{A}_2^*\}$*, let* $\mathsf{cheat}_{\mathrm{X}}$ *denote the event that experiment* X *contains a malformed tuple with respect to* $\boldsymbol{R}_{\mathsf{fac}}, \boldsymbol{R}_{\mathsf{enc\text{-}elg}}, \boldsymbol{R}_{\mathsf{aff\text{-}g}}$*. Furthermore, let* $\widetilde{\boldsymbol{\tau}}_{\mathrm{X}}$ *denote random variable that returns a transcript of* X *conditioned on the event that all relevant tuples are well-formed with respect to* $\boldsymbol{R}_{\mathsf{fac}}, \boldsymbol{R}_{\mathsf{enc\text{-}elg}}, \boldsymbol{R}_{\mathsf{aff\text{-}g}}$*. Then,*

$$\begin{cases} \Pr_{\boldsymbol{\tau}_{\mathrm{A}_1^*}}[\mathsf{cheat}_{\mathrm{A}_1^*}] = \Pr_{\boldsymbol{\tau}_{\mathrm{A}_2^*}}[\mathsf{cheat}_{\mathrm{A}_2^*}] \\ \mathrm{SD}(\widetilde{\boldsymbol{\tau}}_{\mathrm{A}_2^*}, \widetilde{\boldsymbol{\tau}}_{\mathrm{A}_1^*}) = 0 \end{cases}$$

Proof. Consider the following intermediate (inefficient) experiment A_1^{**}: Define $\mathcal{R}_{\mathrm{A}_1^{**}}$ to run the code of $\mathcal{R}_{\mathrm{A}_1^*}$ until a malformed tuple is detected with respect to $\boldsymbol{R}_{\mathsf{fac}}, \boldsymbol{R}_{\mathsf{enc\text{-}elg}}, \boldsymbol{R}_{\mathsf{aff\text{-}g}}$ (i.e. $\mathsf{cheat}_{\mathrm{A}_1^*}$ occurred), and then switch to $\mathcal{R}_{\mathrm{A}_2^*}$ from that point onward. Clearly, $\Pr_{\boldsymbol{\tau}_{\mathrm{A}_1^*}}[\mathsf{cheat}_{\mathrm{A}_1^*}] = \Pr_{\boldsymbol{\tau}_{\mathrm{A}_1^{**}}}[\mathsf{cheat}_{\mathrm{A}_1^{**}}]$ and $\boldsymbol{\tau}_{\mathrm{A}_1^{**}} \equiv \boldsymbol{\tau}_{\mathrm{A}_2^*}$ and $\widetilde{\boldsymbol{\tau}}_{\mathrm{A}_1^{**}} \equiv \widetilde{\boldsymbol{\tau}}_{\mathrm{A}_1^*}$. This concludes the proof. □

Claim 7.15. *It holds that* $\Pr_{\boldsymbol{\tau}_{\mathrm{A}_1}}[\mathsf{cheat}_{\mathrm{A}_1}] \geq \delta - 3d \cdot \nu$.

Proof. Let $\delta^* = \Pr_{\boldsymbol{\tau}_{\mathrm{A}_1^*}}[\mathsf{forge}_{\mathrm{A}_1^*}] - \Pr_{\boldsymbol{\tau}_{\mathrm{A}_2}}[\mathsf{forge}_{\mathrm{A}_2^*}]$ and notice that $\delta^* \geq \delta - 2d\nu$ by Claim 7.14. Thus,

$$\begin{aligned} \delta - 2d\nu &\leq \Pr_{\boldsymbol{\tau}_{\mathrm{A}_1^*}}[\mathsf{forge}_{\mathrm{A}_1^*} \wedge \mathsf{cheat}_{\mathrm{A}_1^*}] - \Pr_{\boldsymbol{\tau}_{\mathrm{A}_2}}[\mathsf{forge}_{\mathrm{A}_2^*} \wedge \mathsf{cheat}_{\mathrm{A}_2^*}] && (2) \\ &\leq \Pr_{\boldsymbol{\tau}_{\mathrm{A}_1^*}}[\mathsf{cheat}_{\mathrm{A}_1^*}] \left(\Pr_{\boldsymbol{\tau}_{\mathrm{A}_2^*}}[\mathsf{forge}_{\mathrm{A}_1^*} \mid \mathsf{cheat}_{\mathrm{A}_1^*}] - \Pr_{\boldsymbol{\tau}_{\mathrm{A}_2^*}}[\mathsf{forge}_{\mathrm{A}_2^*} \mid \mathsf{cheat}_{\mathrm{A}_2^*}] \right) \\ &\leq \Pr_{\boldsymbol{\tau}_{\mathrm{A}_1^*}}[\mathsf{cheat}_{\mathrm{A}_1^*}] \leq \Pr_{\boldsymbol{\tau}_{\mathrm{A}_1}}[\mathsf{cheat}_{\mathrm{A}_1}] + d \cdot \nu && (3) \end{aligned}$$

where Equations (2) and (3) follow from Claim 7.13 and Claim 7.14. To conclude, since experiments A_1 and A_1^*, and experiments A_2 and A_2^*, are statistically $(d \cdot \nu)$-indistinguishable, the claim follows. □

As a final intermediate step, we will be using the generalized forking lemma, stated next.

Lemma 7.16 (Generalized Forking Lemma [4])**.** *Let* $\mathcal{A}$ *denote a PPTM with inputs* $(x, h_1, \ldots, h_d; r)$ *that outputs a pair* $(j, z) \in \{0, 1, \ldots, d\} \times \{0,1\}^*$*, where* r *denotes* $\mathcal{A}$*'s random tape. Let* $\boldsymbol{x}$ *denote denote some arbitrary distribution and let* $\boldsymbol{H}$ *denote a finite set. Let*

$$\delta = \Pr_{\substack{x \leftarrow \boldsymbol{x}, r \leftarrow \{0,1\}^* \\ \{h_i \leftarrow \boldsymbol{H}\}_{i=1}^d}} [(j, z) \leftarrow \mathcal{A}(x, h_1, \ldots, h_d; r) \;\; s.t. \;\; j > 0]$$

and consider the 'forking' algorithm $\mathcal{F}$*:*

1. *Let* $x \leftarrow \boldsymbol{x}$ *and sample* r *according to the prescribed distribution.*
2. *Sample* $\{h_i \leftarrow \boldsymbol{H}\}_{i=1}^d$.
3. *Run* $\mathcal{A}$ *on input* $(x, h_1, \ldots, h_d; r)$*. Obtain* (j, z)*.*

   If $j = 0$*, output* 0.
4. *Sample* $\{h_i' \leftarrow \boldsymbol{H}\}_{i=j}^d$
5. *Run* $\mathcal{A}$ *on input* $(x, h_1, \ldots, h_{j-1}, h_j', \ldots, h_d'; r)$*. Obtain* (j', z')*.*

   If $j = j'$ *and* $h_j \neq h_j'$*, output* 1*. Else, output* 0*.*

Let δ' *denote the probability that* $\mathcal{F}$ *outputs* 1*. Then,* $\delta' \geq \frac{\delta^2}{d} - \frac{1}{|\boldsymbol{H}|}$.

Putting everything together. Apply the forking lemma with $\mathcal{A} = (\mathcal{R}_{\mathrm{A}_1}, \mathcal{Z})$ in experiment A_1, viewed as a single machine, r being the random tape of $\mathcal{R}_{\mathrm{A}_1}$ and $\mathcal{Z}$, viewed as a single random tape, $\boldsymbol{H}$ and $\{h_i\}_{i=1}^d$ being the RO output space and query-answers, $x = (\hat{N}, s, t)$ denotes the Pedersen tuple, and (j, z) to denote the index and the accepting, valid proof of the first malformed tuple included in the transcript; $(j, z) = (0, \perp)$ if no such tuple exists. Using the bound $\varphi(N)/(N-1) - 1/2 \geq 1/3$,[16] Proposition 7.10 follows from Lemma 7.16, Claims 5.1 and 7.15, and Theorem 5.2.

[16] This bound holds true as long as $N^{3/4} \geq p, q$ and $N^{1/4} \geq 12$ for RSA number $N = pq$.

7.2 Reduction to DDH and DCR

7.2.1 Hybrid B_0

FIGURE 19 (Hybrid B_0, i.e. Forgery Experiment B_0)

Define $\mathcal{R}_{\mathrm{B}_0}$ interacting with $\mathcal{Z}$ corrupting all players but $\mathcal{P}_b$.

Sample $(x, X) \leftarrow \mathbb{F}_q \times \mathbb{G}$ such that $X = g^x$. Proceed as $\mathcal{R}_{\mathrm{A}_2}$ except that:

1. *Key-Generation & Aux Info.*

   Same as Figure 18.

2. *Presigning.* Sample $\gamma, \delta \leftarrow \mathbb{F}_q$ and set $\Gamma = g^\gamma$.
   (a) Sample $(Y_b, A_{b,1}, A_{b,2}, B_{b,1}, B_{b,2}) \leftarrow \mathbb{G}^5$.
   (b) Set $\Gamma_b = \Gamma \cdot (\prod_{j \neq b} g^{\gamma_j})^{-1}$ and $\delta_b = \delta - (\sum_{j \neq b} k_j) \cdot (\sum_{j \neq b} \gamma_j) + \sum_{j \neq b} \alpha_{j,b} + \beta_{j,b} \mod q$
   (c) Set $\Delta_b = g^\delta \cdot \prod_{j=1}^n \Gamma^{-k_j}$ and $S_b = X^\delta \cdot \Gamma^{-(\sum_{j \neq b} k_j) \cdot (\sum_{j \neq b} x_j) + \sum_{j \neq b} \hat{\alpha}_{j,b} + \hat{\beta}_{j,b}}$
   (d) Set all other items as in Figure 18.

3. *Signing.* Set $\sigma = \gamma^{-1} \cdot (m + r \cdot x) \mod q$ and $r = \Gamma|_{(\cdot)}$.
   (a) Set $\sigma_b = \sigma - (\sum_{j \neq b} k_j)/\delta \cdot m - r \cdot \left((\sum_{j \neq b} k_j) \cdot (\sum_{j \neq b} x_j) + \sum_{j \neq b} \hat{\alpha}_{j,b} + \hat{\beta}_{j,b} \right) / \delta \mod q$
   (b) Set all other items as in Figure 18.

Output. Same as $\mathcal{R}_{\mathrm{A}_2}$ in Figure 18.

Figure 19: Hybrid B_0, i.e. Forgery Experiment B_0

Proposition 7.17. *Let ν denote smallest (statistical) soundness parameter in the protocol and let d denote an upper bound on the number of of RO queries. Let α denote the probability that $\mathcal{Z}$ outputs a forgery in Experiment A_2 from Figure 18 and assume that $\mathcal{Z}$ outputs a forgery with probability less than $\alpha - \delta$ in Experiment B_0 from Figure 19. Then, there exists efficient PPTM $\mathcal{A}$ such that:*

$$\Pr_{(A,B,C,z) \leftarrow \mathsf{DDH}(1^\kappa)} [\beta \leftarrow \mathcal{A}(1^\kappa, A, B, C) \ \ s.t. \ \ \beta = z] \geq \frac{1}{2} + \delta/2 - \nu \cdot d$$

7.2.2 Proof of Proposition 7.17

Define in-between experiment A_2^* relative to Item 2a of Figure 19: $\mathcal{R}_{\mathrm{A}_2^*}$ proceeds as $\mathcal{R}_{\mathrm{B}_0}$ except that instead of sampling the tuple $(Y_b, A_{b,1}, A_{b,2}, B_{b,1}, B_{b,2})$ randomly, it calculates the tuple as in experiment A_2 from Figure 18. Let α, α^* and α_0 denote the probability that $\mathcal{Z}$ outputs a forgery in experiment A_2, A_2^* and B_0 respectively. It is assumed that $\alpha - \alpha_0 \geq \delta$ for $\delta \geq 0$.

Claim 7.18. *Recall that ν denotes smallest (statistical) soundness parameter in the protocol and let d denote an upper bound on the number of of RO queries. Using the notation above, it holds that $\alpha^* \geq \alpha - \nu \cdot d$.*

Proof. Recall that Π^{elog} is statically ν_0-sound for $\nu_0 \geq \nu$. The claim is immediate by noting that A_2^* and A_2 are identically distributed if all $(\Delta_j, \Gamma, A_{j,1}, A_{j,2}, Y_j, g)$ are well-formed, i.e. $(\Delta_j, \Gamma, A_{j,1}, A_{j,2}, Y_j, g) \in \boldsymbol{R}_{\mathsf{elog}}$. □

Definition 7.19. For $(A, B, C) \in \mathbb{G}^3$, define the following 'variant' of DH multi-commitment. To 'commit' to $k_1, k_2 \in \mathbb{F}_q$, define randomized algorithm $\widetilde{\mathsf{com}}_{A,B,C}(k_1, k_2)$ as follows:

1. Set $(A_0, B_0, C_0) = (A \cdot g^\mu, B^\lambda g^\rho, A^\rho B_0^\mu C^\lambda)$ for $\mu, \lambda, \rho \leftarrow \mathbb{F}_q$.
2. Return $(A_0, B_0, C_0 \cdot g^{k_1}, B_0^u g^v, C_0^u A_0^v \cdot g^{k_2})$ for $u, v \leftarrow \mathbb{F}_q$.

Reduction to DDH. Define algorithm $\mathcal{R}_{\mathsf{DDH}}$ taking input $(A, B, C) \in \mathbb{G}^3$: Run $\mathcal{R}_{\mathrm{A}_2^*}$ but instead of generating the tuple $(Y_b, A_{b,1}, A_{b,2}, B_{b,1}, B_{b,2})$ as prescribed, use the 'special' commitment process $\widetilde{\mathsf{com}}_{A,B,C}(\cdot)$ from Definition 7.19 to commit to a pair (k, γ). Output 0 if the execution yields a forgery and 1 otherwise. Proposition 7.17 follows from the claim below.

Claim 7.20. *It holds that* $\Pr_{(A,B,C,z)\leftarrow\mathsf{DDH}(1^\kappa)}[\beta \leftarrow \mathcal{R}_{\mathsf{DDH}}(A, B, C) \;\; s.t. \;\; \beta = z] \geq \frac{1}{2} + (\delta - d \cdot \nu)/2$

Proof. The first thing to note is that $(A_0, b_0, C_0, B_0', C_0') = \widetilde{\mathsf{com}}_{A,B,C}(k_1, k_2)$ is either a random 5-tuple when A, B, C is a random 3-tuple, or it is identically distributed with the 5-tuple $(Y_b, A_{b,1}, A_{b,2}, B_{b,1}, B_{b,2})$ (i.e. it is a commitment to k_1, k_2). Thus, $\mathcal{R}_{\mathsf{DDH}}$ corresponds to experiment A_2 when A, B, C is a DH tuple, or it corresponds to B_0 when A, B, C is a random 3-tuple. Finally, since $\mathcal{R}_{\mathsf{DDH}}$ is 'correct' (outputs the right bit) when (A, B, C) is a DH tuple and the execution yields a forgery, or when (A, B, C) is a random 3-tuple and the execution does not yield a forgery, we conclude that:

$$\begin{aligned}\Pr[\beta \leftarrow \mathcal{R}_{\mathsf{DDH}}(A, B, C) \;\text{ s.t. }\; \beta = z] &= \frac{1}{2} \cdot \alpha_0^* + \frac{1}{2} \cdot (1 - \alpha_0) = \frac{1}{2} + \frac{\alpha_0^* - \alpha_0}{2} \\ &\geq \frac{1}{2} + \frac{\alpha - \alpha_0 - d \cdot \nu}{2} \geq \frac{1}{2} + \frac{\delta - d \cdot \nu}{2}\end{aligned}$$

□

7.2.3 Hybrid B_1

FIGURE 20 (Hybrid B_1, i.e. Forgery Experiment B_1)

Define $\mathcal{R}_{\mathrm{B}_1}$ interacting with $\mathcal{Z}$ corrupting all players but $\mathcal{P}_b$.

Sample $(x, X) \leftarrow \mathbb{F}_q \times \mathbb{G}$ such that $X = g^x$. Proceed as $\mathcal{R}_{\mathrm{B}_0}$ except that:

1. *Key-Generation & Aux-info.*

   Same as Figure 19.

2. *Presigning.*
   (a) Set $K_b, G_b, \{F_{j,b}, \hat{F}_{j,b}\}_{j \neq b}$ as random elements in $\mathbb{Z}_{N_b^2}^*$.
   (b) Set all other items as in Figure 19.

Output. Same as $\mathcal{R}_{\mathrm{B}_0}$ in Figure 19.

Figure 20: Hybrid B_1, i.e. Forgery Experiment B_1

Proposition 7.21. *Let α denote the probability that $\mathcal{Z}$ outputs a forgery in Experiment B_0 from Figure 19 and assume that $\mathcal{Z}$ outputs a forgery with probability less than $\alpha - \delta$ in Experiment B_1 from Figure 20. Let d denote an upper bound on the number of key-refresh epochs. Then, there exists efficient PPTM $\mathcal{A}$ such that:*

$$\Pr_{(N,C,z)\leftarrow\mathsf{DCR}(1^\kappa)}[\beta \leftarrow \mathcal{A}(1^\kappa, N, C) \;\; s.t. \;\; \beta = z] \geq \frac{1}{2} + \delta/2d$$

7.2.4 Proof of Proposition 7.21

Next, let $\mathrm{B}_1^{(0)} = \mathrm{B}_0$ and define in-between experiments $\mathrm{B}_1^{(1)}, \ldots, \mathrm{B}_1^{(d)} = \mathrm{B}_1$ with respect to each key-refresh epoch of Figure 20. In these experiments, $\mathcal{R}_{\mathrm{B}_1^{(i)}}$ proceeds as $\mathcal{R}_{\mathrm{B}_0}$ up to and including the $(d-i)$-th key-refresh epoch, and as $\mathcal{R}_{\mathrm{B}_1}$ from the $(d-i+1)$-the key-refresh epoch until the end. Let $\alpha^{(i)}$ denote the probability that $\mathcal{Z}$ outputs a forgery in experiment $\mathrm{B}_1^{(i)}$.

Claim 7.22. *There exists $i \in [d]$ such that $\alpha^{(i)} - \alpha^{(i-1)} \geq \delta/d$.*

Proof. Simple averaging argument. □

Definition 7.23. For $(N, C) \in \mathbb{N} \times \mathbb{Z}_{N^2}^*$, define the following 'variant' of Paillier encryption: to 'encrypt' $k \in \mathbb{Z}_N$, return $\widetilde{\mathsf{enc}}_{N,C}(k) = (1 + k \cdot N) \cdot \lambda^N \cdot C^\lambda \mod N^2$ for $\lambda, \rho \leftarrow \mathbb{Z}_N^*$.

Reduction to DCR. Define algorithm $\mathcal{R}_{\mathsf{DCR}}$ that takes as input $(N, C) \in \mathbb{N} \times \mathbb{Z}^*_{N^2}$. Run $\mathcal{R}_{\mathrm{B}_0}$ up to the $(d-i)$-th epoch. For the next epoch, in the main path, use $N_b = N$ and the special Paillier encryption process $\widetilde{\mathsf{enc}}_{N,C}(\cdot)$ from Definition 7.23 to encrypt elements under N_b, while in the forked path, proceed as prescribed by $\mathcal{R}_{\mathrm{B}_0}$. For the subsequent epochs, run $\mathcal{R}_{\mathrm{B}_1}$. Output 0 if the execution yields a forgery and 1 otherwise. Proposition 7.21 follows from the claim below.

Claim 7.24. *It holds that* $\Pr_{(N,C,z)\leftarrow\mathsf{DCR}(1^\kappa)}[\beta \leftarrow \mathcal{R}_{\mathsf{DCR}}(N, C) \;\; s.t. \;\; \beta = z] \geq \frac{1}{2} + \delta/2d$

Proof. The first thing to note is that $C_0 = \widetilde{\mathsf{enc}}_{N,C}(k)$ is either a random value in $\mathbb{Z}^*_{N^2}$ when $C \leftarrow \mathbb{Z}^*_{N^2}$, or it is identically distributed with $\mathsf{enc}_N(k)$ (i.e. it is an encryption of k). Thus, $\mathcal{R}_{\mathsf{DCR}}$ corresponds to experiment $\mathrm{B}_1^{(i-1)}$ when C is an encryption of zero, or it corresponds to $\mathrm{B}_1^{(i)}$ when C is a random element in $\mathbb{Z}^*_{N^2}$. Finally, since $\mathcal{R}_{\mathsf{DCR}}$ is 'correct' (outputs the right bit) when $C = \mathsf{enc}_N(0)$ and the execution yields a forgery, or when $C \leftarrow \mathbb{Z}^*_{N^2}$ and the execution does not yield a forgery, we conclude that:

$$\Pr[\beta \leftarrow \mathcal{R}_{\mathsf{DCR}}(N, C) \;\text{ s.t. }\; \beta = z] = \frac{1}{2} \cdot \alpha^{(i)} + \frac{1}{2} \cdot (1 - \alpha^{(i-1)}) \geq \frac{1}{2} + \delta/2d$$

□

7.3 Reduction to Enhanced Existential Unforgeability of ECDSA

FIGURE 21 (Hybrid C, i.e. Forgery Experiment C)

Define $\mathcal{R}_{\mathrm{C}}$ interacting with $\mathcal{Z}$ corrupting all players but $\mathcal{P}_b$.

Call the signing oracle $\mathcal{G}$ obtain ECDSA pk X. Proceed as $\mathcal{R}_{\mathrm{B}_1}$ except that:

1. *Key-Generation & Aux Info.*

   Same as Figure 20 except that $x_b = \perp$.

2. *Presigning.*
   (a) Sample $\delta \leftarrow \mathbb{F}_q$ and call the signing oracle $\mathcal{G}$ to obtain ECDSA presignature Γ.
   (b) Set $\{D_{j,b} = \mathsf{enc}_j(\alpha_{j,b}), \hat{D}_{j,b} = \mathsf{enc}_j(\hat{\alpha}_{j,b})\}_{j\neq b}$ for $\alpha_{j,b}, \hat{\alpha}_{j,b} \leftarrow \boldsymbol{J}$
   (c) Set all other items as in Figure 20.

3. *Signing.*
   (a) Call the signing oracle $\mathcal{G}$ on presignature Γ and message msg. Obtain signature string σ.
   (b) Set σ_b as in Figure 20.

Output. Same as $\mathcal{R}_{\mathrm{B}_1}$ in Figure 20.

Figure 21: Hybrid C, i.e. Forgery Experiment C

Proposition 7.25. *Define random variables* $\boldsymbol{a}$ *and* $\hat{\boldsymbol{a}}_k$ *such that* $\alpha \leftarrow \boldsymbol{a}$ *for* $\alpha \leftarrow \boldsymbol{J}$ *and* $k' + \alpha \leftarrow \hat{\boldsymbol{a}}$ *for* $\alpha \leftarrow \boldsymbol{J}$. *Let* $\nu = \max_{k' \in \boldsymbol{I}_{\ell+\varepsilon}}(\mathrm{SD}(\boldsymbol{a}, \boldsymbol{a}_k))$. *Let* α *denote the probability that* $\mathcal{Z}$ *outputs a forgery in Experiment* B_1 *from Figure 20 and let* d *denote a bound on the number of presignatures. It holds that* $\mathcal{Z}$ *outputs a forgery with probability* $\alpha - 2dn \cdot \nu$ *in Experiment* C *from Figure 21.*

Proof. The proof is immediate by the definition of ν, since $\mathcal{R}_{\mathrm{C}}$ generates d samples of $\{D_{j,b}, \hat{D}_{j,b}\}_{j\neq b}$. □

7.3.1 Dealing with attackers that do not abort

At the beginning of this section, we assumed that whenever $\mathcal{Z}$ disrupts the key-refresh (specifically by preventing parties from obtaining output), the entire protocol was halted. However, as noted in Remark 4.2, the attacker is actually allowed to continue signature operations even after disrupting the key-refresh (with the key-refresh not being invoked again). We refer to attackers who abort the protocol whenever they disrupt the key-refresh as *aborting* attackers.

In this section, we prove a reduction from non-aborting to aborting attackers, as captured by the following lemma:

Lemma 7.26. *For any $\mathcal{Z}$ that forges signatures with probability α in an execution of Σ, there exists an aborting attacker $\mathcal{Z}_0$ that forges signatures with probability α/d, where $d-1$ denotes an upper bound on the number of key refresh epochs.*

Proof. We define $\mathcal{Z}_0$ with black-box access to $\mathcal{Z}$ as follows: $\mathcal{Z}_0$ samples $j \leftarrow [d]$ and acts as an intermediary between $\mathcal{Z}$ and the honest parties up to and including epoch $i-1$. At the key-refresh initiating epoch i (if $i = d$, then this step is irrelevant), $\mathcal{Z}_0$ impersonates the honest parties and generates all the auxiliary material (Paillier keys, Pedersen keys, DH tuples, etc.). If $\mathcal{Z}$ disrupts the key-refresh but does not abort, $\mathcal{Z}_0$ reverts to acting as an intermediary between the honest parties and $\mathcal{Z}$—this is possible because the epoch counter has not been incremented, and the auxiliary material has not been updated in $\mathcal{Z}$'s view. If $\mathcal{Z}$ does not disrupt the key-refresh, the experiment is considered a failure, and $\mathcal{Z}_0$ outputs an error and halts. Otherwise, $\mathcal{Z}_0$ outputs whatever $\mathcal{Z}$ outputs and then halts.

Notice that with probability $1/d$ (i.e. when $\mathcal{Z}_0$ correctly guesses the cheating round), $\mathcal{Z}$'s view in the experiment described above is identically distributed to a random execution of Σ in the presence of $\mathcal{Z}$. As a result, $\mathcal{Z}_0$ outputs a forgery with probability α/d, where α denotes $\mathcal{Z}$'s forgery probability. □

8 How to handle Adaptive Corruptions

Previously, we assumed that $\mathcal{Z}$ corrupts all-but-one parties statically at the beginning of the execution. Since the protocol is completely 'symmetric', meaning there is no way to distinguish parties other than by their arbitrarily-assigned labels in $\boldsymbol{P}$, there is a simple, intuitive way to adapt the above to the adaptive corruption model. This can be done by randomly assigning $\mathcal{P}_b$ to be one of the honest parties at the beginning of each key-refresh epoch of each experiment, and resetting the experiment back to the beginning of the key-refresh epoch if $\mathcal{Z}$ decides to corrupt $\mathcal{P}_b$ during that epoch.

The above-described process impacts the forgery probability only if the running time of the experiment exceeds some predefined polynomial bound. As long as the new experiment terminates within this bound, the view of $\mathcal{Z}$ in the new experiment is identically distributed with the view of $\mathcal{Z}$ in the relevant experiment, up to a relabeling of the (simulated) honest parties' identities between epochs.

Adaptive to Static Security Reduction. Formally, we prove a security reduction from the adaptive to the static corruption model. By increasing the number of key-refresh epochs, we show that any adaptive forger in Σ can be transformed into a static forger, with a negligible loss in the forgery probability.

The main challenge for the reduction is to 'break' the sequence of public key shares $X_b \to X_b' \to \ldots \to X_b^{(d)}$, where $X_b^{(i)}$ denotes the sole honest party's (say $\mathcal{P}_b$) public key share in epoch d. The goal is to ensure that, from $\mathcal{Z}$'s perspective, $X_b^{(i)}$ and $X_b^{(i+1)}$ belong to the same party with probability $1/h$ when there are at least h honest parties at the beginning of epoch $i+1$. Before describing the reduction, we present the formal security claim expressed in asymptotic terms as a function of the security parameter κ.

Lemma 8.1. *Recall that κ denotes the security parameter. Let $\widetilde{\mathcal{Z}}$ denote a PPTM that adaptively corrupts parties in an execution of Σ with at most d key-refresh epochs, and assume that $\widetilde{\mathcal{Z}}$ outputs a forgery with probability α. Recall that n denotes the number of parties in Σ. Then, the following holds under DDH for some negligible function μ. For any $t \in \omega(\log(\kappa))$, there exists a PPTM $\mathcal{Z}$ that statically corrupts all-but-one parties in an execution of Σ with at most $d \cdot tn$ key-refresh epochs such that $\mathcal{Z}$ outputs a forgery with probability at least $\alpha - \mu$.*

8.1 Proof of Lemma 8.1

Similar to the proof of Theorem 6.6, we present a sequence of inbetween hybrid experiments which are indistinguishable from each other, and the last experiment corresponds to the static corruption case. Given the similarity of the security proofs in this section to those in Section 7, we will keep the exposition concise. Hereafter, the security parameter and all the dependent variables are fixed.

Experiment A. Let $\boldsymbol{C}$ denote the set of parties that are not corrupted by $\widetilde{\mathcal{Z}}$ at any given time in an execution of Σ. This set evolves dynamically as the execution progresses. Define algorithm $\mathcal{R}_{\mathrm{A}}$ that runs the

code of the honest parties against $\mathcal{Z}$ with the following additional instruction: At the beginning of each key-refresh epoch, $\mathcal{R}_{\mathrm{A}}$ randomly chooses a 'special' party $\mathcal{P}_b \leftarrow \boldsymbol{P} \setminus \boldsymbol{C}$. If $\mathcal{Z}$ decides to corrupt $\mathcal{P}_b$, the experiment resets to the beginning of the last key-refresh epoch and a new special party is chosen. If this step occurs more than $t \cdot n$ times in any given epoch of $\widetilde{\Sigma}$, $\mathcal{Z}$ is instructed to halt. If $\mathcal{Z}$ decides to corrupt a party other than $\mathcal{P}_b$, the experiment continues, with $\mathcal{R}_{\mathrm{A}}$ revealing that party's state to $\mathcal{Z}$.

Claim 8.2. *If $\widetilde{\mathcal{Z}}$ outputs a forgery with probability α in an execution of Σ, then $\mathcal{R}_{\mathrm{A}}$ outputs a forgery with probabilty $\alpha - d\exp(-t)$.*

Proof. Clearly, if $\mathcal{R}_{\mathrm{A}}$ does not halt prematurely, then the view of $\widetilde{\mathcal{Z}}$ an execution of Σ and experiment $\mathcal{A}$ is identically distributed. Thus, it suffices to bound the probability that $\mathcal{R}_{\mathrm{A}}$ aborts prematurely. Since the corruption event of $\mathcal{P}_b$ in any given epoch can be modeled as a sequence of $t \cdot n$ Bernoulli trials with probability $1/n$ (because the probability of the 'good' event, where $\widetilde{\mathcal{Z}}$ does not guess $\mathcal{P}_b$, is at least $\frac{1}{|\boldsymbol{P} \setminus \boldsymbol{C}|}$), Claim 8.2 follows from the fact below. □

Fact 8.3. *For $t, n \in \mathbb{N}$, let $r = t \cdot n$ and define $\boldsymbol{\sigma} = \sum_{i=1}^{n} \boldsymbol{y}_i$ for $\boldsymbol{y}_1 \ldots \boldsymbol{y}_r$ iid Boolean variables with $\Pr[\boldsymbol{y}_i = 1] = 1/n$. It holds that $\Pr[\boldsymbol{\sigma} = 0] \leq \exp(-t)$.*

Proof. Notice that $\Pr[\boldsymbol{\sigma} = 0] = \left(\frac{n-1}{n}\right)^r$. Using the fact that $(1 + x/m)^m \leq \exp(x)$ for $m \geq 1$ and $|x| \leq m$:

$$\left(\frac{n-1}{n}\right)^r = \left(1 - \frac{1}{n}\right)^r = \left(1 + \frac{-t}{r}\right)^r \leq \exp(-t)$$

□

8.1.1 Reduction to DDH

Experiment B. Define algorithm $\mathcal{R}_{\mathrm{B}}$ that proceeds as $\mathcal{R}_{\mathrm{A}}$ with the following additional instructions. Let $\mathcal{P}_b$ and $\mathcal{P}_{b'}$ denote the chosen special parties in epochs $i-1$ and i, respectively. At the beginning of each epoch i, proceed as follows: Instead of calculating $C_{b',b} = x_{b,b'} + \rho_{b,b'} \mod q$, where $\rho_{b,b'}$ is derived from the DH tuple using the oracle $\mathcal{H}$ and $x_{b,b'}$ is the random shift sent from $\mathcal{P}_b$ to $\mathcal{P}_{b'}$, set $C_{b',b} \leftarrow \mathbb{F}_q$ instead.

Claim 8.4. *Let δ denote the difference in the probabilities that $\mathcal{R}_{\mathrm{A}}$ and $\mathcal{R}_{\mathrm{B}}$ yield a forgery in their respective experiments. Then, there exists an efficient PPTM $\mathcal{A}$ such that:*

$$\Pr_{(A,B,C,z) \leftarrow \mathsf{DDH}(1^\kappa)} [\beta \leftarrow \mathcal{A}(1^\kappa, A, B, C) \mid \beta = z] \geq \frac{1}{2} + \frac{\delta}{2}.$$

8.1.2 Reduction to the Static Corruption Model

Experiment C. We define adversary $\mathcal{Z}$, which corrupts all parties except $\mathcal{P}_b \in \boldsymbol{P}$, and maintains a simulated execution of Σ with $\widetilde{\mathcal{Z}}$. To avoid confusion, we write $\widetilde{\Sigma}$ for the protocol execution from $\widetilde{\mathcal{Z}}$'s point of view, and values in $\widetilde{\Sigma}$ are generally denoted with a tilde.

1. $\mathcal{Z}$ locally emulates all parties except those in $\boldsymbol{C} \cup \{\mathcal{P}_b\}$.
2. $\mathcal{Z}$ acts as an intermediary between $\widetilde{\mathcal{Z}}$, and $\mathcal{P}_b$ and $\mathcal{H}$, possibly relabeling some of the message identifiers but not the actual messages, except for those described in Item 5. $\mathcal{Z}$ also keeps track of the random-oracle queries, ensuring the relabeling of the message identifiers is consistent with the oracle from $\widetilde{\mathcal{Z}}$'s perspective. This gives rise to a simulated random oracle, $\widetilde{\mathcal{H}}$, which $\mathcal{Z}$ maintains locally.
3. At the beginning of each key-refresh epoch, $\mathcal{Z}$ re-randomizes the identity of $\mathcal{P}_b$ swithing the identifying labels between $\mathcal{P}_b$ and a random party in $\boldsymbol{P} \setminus \boldsymbol{C}$ (including $\mathcal{P}_b$) in $\widetilde{\Sigma}$. The execution then continues as per Items 1 and 2.
4. If $\widetilde{\mathcal{Z}}$ decides to corrupt in $\widetilde{\Sigma}$ the party corresponding to $\mathcal{P}_b$ in Σ, a new key-refresh epoch is initiated in Σ and $\widetilde{\mathcal{Z}}$ is rewound back to the last key-refresh epoch of $\widetilde{\Sigma}$. If this step occurs more than $t \cdot n$ times in any given epoch of $\widetilde{\Sigma}$, $\mathcal{Z}$ is instructed to halt.

5. In every key refresh execution of $\widetilde{\Sigma}$, for the tuple of public shifts (i.e. the $X_{i,j}$'s), $\mathcal{Z}$ hands over $(\widetilde{X}_{b,1}, \ldots, \widetilde{X}_{b,n})$ to $\widetilde{\mathcal{Z}}$ on behalf of $\mathcal{P}_b$ instead of $(X_{b,1}, \ldots, X_{b,n})$. Here, $(\widetilde{X}_{b,b}, \widetilde{X}_{b,b'}) = (X_{b,b} \cdot \Delta, X_{b,b'} \cdot \Delta^{-1}$, where $\mathcal{P}_{b'}$ is the chosen identity for $\mathcal{P}_b$ for the new epoch, and Δ is the difference between the prescribed public key shares for $\mathcal{P}_b$ and $\mathcal{P}_{b'}$ at the completion of the key refresh.[17] All other values are set in the expected way. Note that the entries of $(\widetilde{X}_{b,1}, \ldots, \widetilde{X}_{b,n})$ are sorted according to the ongoing labeling of the parties from $\widetilde{\mathcal{Z}}$'s point of view. Additionally,
   (a) $\mathcal{Z}$ generates proofs for the pair $(\widetilde{X}_{b,b}, \widetilde{X}_{b,b'})$ on behalf of $\mathcal{P}_b$ using the ZK simulator for Π^{sch}.
   (b) The ciphertext $\widetilde{C}_{b,b'}$ in $\widetilde{\Sigma}$ sent from $\mathcal{P}_b$ to $\mathcal{P}_{b'}$ is chosen randomly from $\mathbb{F}_q$.
6. At the end of the execution, if $\mathcal{Z}$ did not halt prematurely, then $\mathcal{Z}$ outputs whatever $\widetilde{\mathcal{Z}}$ and halts.

Claim 8.5. *If experiment* B *yields a forgery with probability α, then static adversary $\mathcal{Z}$ corrupting all but-one-parties outputs a forgery with probability α in an execution of Σ (i.e. Experiment* C*).*

Proof. From $\widetilde{\mathcal{Z}}$ perspective, since the ZK simulator for Π^{sch} is perfect, it follows that, from $\widetilde{\mathcal{Z}}$'s perspective, experiments C and B are identically distributed. □

9 Additional Related Work

Prior works to [21] on threshold ECDSA. All previous protocols for threshold ECDSA follow (variants) of the blueprint described in Section 2.1 where the parties locally generate shares $\gamma_1^* \ldots \gamma_n^*$ of γ [52, 34] or γ^{-1} [38, 24] and then jointly compute $r = g^{\gamma}|_{(\cdot)}$ and shares of $\gamma^{-1}(m + rx)$ 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; this is the main source of the round-complexity cost.

Security models in prior works. As mentioned previously, Gennaro and Goldfeder [38] show that their protocol satisfies a game-based definition of security (i.e. unforgeability) under standard assumptions (DDH, DCR, strong-RSA, ECDSA). The protocol of Castagnos et al. [24] follows the same template, except that it replaces Paillier with an encryption scheme based on class groups [23]. Specifically, they show that their 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.[18]

Lindell et al. [52] and Doerner et al. [34] show secure-function evaluation of the ECDSA functionality and prove that their respective protocols UC-realize said functionality in a hybrid model with ideal commitments and zero-knowledge, assuming DDH .

On the GG20 and CMP protocols. The independent works of Gennaro and Goldfeder [39], known as 'GG20', and Canetti et al. [22], known as 'CMP', are precursors to [21] and the present work.[19] These works were submitted independently to the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS'20). On the authors' initiative, the two works were merged, resulting in [21], which includes two distinct non-interactive signing protocols offering a trade-off between the presigning round complexity and the general resource complexity for achieving accountability. The protocol presented in this document is the culmination of several improvements and simplifications, achieving the best costs compared to the two protocols in [21].

[17] We note that this is possible because $\mathcal{Z}$ has access to the RO queries of $\widetilde{\mathcal{Z}}$ (as $\mathcal{Z}$ acts as an intermediary between $\widetilde{\mathcal{Z}}$ and the RO) and, being a rushing adversary, $\mathcal{Z}$ also has access to $\mathcal{P}_b$'s public shifts before handing them over to $\widetilde{\mathcal{Z}}$. Consequently, $\mathcal{Z}$ can compute the Δ without issue.

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

[19] [39] and [22] are available in the IACR eprint archive.

Concurrent works with [21]. The works of [39, 22], Damgård et al. [30], and Gągol and Straszak [44] were published on the IACR eprint archive within a few days of each other. Interestingly, all of these works mention a pre-processing mode of operation to improve efficiency. In [30], the authors consider the *honest-majority* setting and design a protocol based on Gennaro et al. [40]. They show that their protocol is UC-secure with abort and also demonstrate how to bootstrap their protocol to achieve fairness. Additionally, they mention a non-interactive variant of their protocol by pre-processing all-but-one of the rounds, though no security analysis is provided for this variant.

In [44], motivated by the application of MPC wallets with a *large* number of signers, the authors design a protocol that supports robustness in the form of identifiable abort by augmenting the protocol in [52] with additional NIZKs.

Bootstrapping authentication for proactive security. Kondi et al. [48] consider the case where some parties remain offline during a proactive refresh phase, and furthermore do not have reliable authenticated communication with the other parties when they get back online. (Indeed, in the context of cryptocurrency custody, it may be desirable for offline 'cold' wallets to participate in the refresh at their own pace, which, in turn, may open the door to attacks.) They show how such a late party can regain authenticated communication with the reset of the system by way of using the blockchain itself as a means to authenticate the public keys of the other parties. This solution can be seen as a way to use the blockchain for implementing the persistent threshold signature scheme in the Canetti et al. [19] solution.

9.1 Works Subsequent to [21]

Given the popularity of ECDSA in the blockchain space, many follow-up and related works have emerged since [21]. Below, we provide a non-exhaustive list of some of the most relevant subsequent works and direct the interested reader to the respective papers for further references.

In addition to using OT and Paillier, numerous works have introduced alternative building blocks to design ECDSA protocols. For instance, the work of Xue et al. [63] shows how to improve the share-computation, typically implemented with Paillier in this work and many others, by using the Joye-Libert cryptosystem instead. The work of Abram et al. [1] shows how to achieve a bandwidth-efficient protocol based on pseudo-random correlation generators (PCGs). Additionally, the work of Boneh et al. [8] introduces the concept of an exponent-VRF, enabling the design of a two-round two-party ECDSA protocol based on standard assumptions.

On Blokh et al. [5]. The work of Blokh et al. [5] extends the present work in several directions. First, Blokh et al. generalize the security paradigm from this work by showing that any signature protocol that satisfies a weaker version of 'standalone' security UC-realizes functionality $\mathcal{F}_{\mathsf{tsig}}$.[20] Additionally, [5] shows how the range proofs herein can be *batched*, meaning that a single proof yields the validity of many ciphertexts, and how these ciphertexts can be *packed*, meaning that each ciphertext contains many signatures. Finally, using the aforementioned tools, the authors design an 'asymmetric' ECDSA protocol for the cold-wallet use case. A similar approach was used in the work of Friedman et al. [36].

Three-Round Threshold ECDSA. The protocol of Doerner et al. [35] was the first to achieve three-round threshold ECDSA in the fully malicious model (all-but-one corruptions). They use a clever technique from Abram et al. [1] and Gennaro et al. [40] that allows for the nonce and the signature shares to be verified together via simple signature verification. This technique is also used in this paper, resulting in either the online three-round protocol or the two-round presigning protocol mentioned in this document. In the latter case, to ensure accountability, the transcript is saved for the non-interactive signing phase.

Presignatures. The work of Shoup and Groth [61] presents a fine-grained analysis of the presignatures mode of operation, where the authors confirm and expand the analysis in the GGM found in Appendix E. Additionally, the authors provide an analysis in the related-key setting, where a number of public keys have a known discrete logarithm relation for the adversary, as well as the so-called 'raw signing' setting. The authors

[20] So-called 'standalone' security for an MPC protocol refers to the security notion where every adversary can be simulated with negligible distinguishing advantage. In [5], the authors demonstrate that it suffices for the protocol to be simulatable with a small-but-not-necessarily-negligible distinguishing advantage in order for the protocol to realize $\mathcal{F}_{\mathsf{tsig}}$.

show various security bounds in the GGM for each of these cases, and they also propose algorithmic techniques to achieve better bounds.

Attacks on Pallier-based threshold ECDSA. The work of Makriyannis et al. [54] presents attacks on several implementations and protocols of Paillier-based threshold ECDSA. The authors exploit various missing checks, either due to deviations from the protocol specifications in certain implementations or because the well-formedness of the Paillier key was not enforced by the protocol specification. None of the attacks from [54] apply here.

Acknowledgements

A previous iteration of this paper contained an inaccurate security bound for enhanced unforgeability in the GGM (cf. Appendix E). We thank Victor Shoup for pointing this out and for further discussions on enhanced unforgeability.

We also thank Omer Shlomovits for identifying an issue regarding the identifiable abort in a previous iteration of this paper. Finally, we thank Geoffroy Couteau for providing valuable feedback on an earlier draft of this manuscript.

References

[1] D. Abram, A. Nof, C. Orlandi, P. Scholl, and O. Shlomovits. Low-bandwidth threshold ecdsa via pseudo-random correlation generators. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2554–2572. IEEE, 2022. doi: 10.1109/SP46214.2022.9833685.

[2] C. Badertscher, R. Canetti, J. Hesse, B. Tackmann, and V. Zikas. Universal composition with global subroutines: Capturing global setup within plain UC. In R. Pass and K. Pietrzak, editors, *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part III*, volume 12552 of *Lecture Notes in Computer Science*, pages 1–30. Springer, 2020. doi: 10.1007/978-3-030-64381-2_1. URL https://doi.org/10.1007/978-3-030-64381-2_1.

[3] E. Barker. Recommendation for key management: Part 1 – general. NIST Special Publication 800-57 Part 1 Rev. 5, National Institute of Standards and Technology, May 2020. URL https://doi.org/10.6028/NIST.SP.800-57pt1r5.

[4] M. Bellare and G. Neven. Multi-signatures in the plain public-key model and a general forking lemma. In A. Juels, R. N. Wright, and S. D. C. di Vimercati, editors, *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, October 30 - November 3, 2006*, pages 390–399. ACM, 2006. doi: 10.1145/1180405.1180453.

[5] C. Blokh, N. Makriyannis, and U. Peled. Efficient asymmetric threshold ECDSA for MPC-based cold storage. Cryptology ePrint Archive, Paper 2022/1296, 2022.

[6] 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.

[7] 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.

[8] D. Boneh, I. Haitner, and Y. Lindell. Exponent-VRFs and their applications. Cryptology ePrint Archive, Paper 2024/397, 2024. https://eprint.iacr.org/2024/397.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. *IACR Cryptology ePrint Archive*, 2000:67, 2020. URL http://eprint.iacr.org/2000/067.

[16] R. Canetti. Universally composable security. *J. ACM*, 67(5), sep 2020. ISSN 0004-5411. doi: 10.1145/3402457. URL https://doi.org/10.1145/3402457.

[17] 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.

[18] 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.

[19] R. Canetti, S. Halevi, and A. Herzberg. Maintaining authenticated communication in the presence of break-ins. *J. Cryptology*, 13(1):61–105, 2000. doi: 10.1007/s001459910004. URL https://doi.org/10.1007/s001459910004.

[20] 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.

[21] R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In J. Ligatti, X. Ou, J. Katz, and G. Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 1769–1787. ACM, 2020. doi: 10.1145/3372297.3423367. URL https://doi.org/10.1145/3372297.3423367.

[22] R. Canetti, N. Makriyannis, and U. Peled. Uc non-interactive, proactive, threshold ecdsa. Cryptology ePrint Archive, Report 2020/492, 2020. https://eprint.iacr.org/2020/492.

[23] 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.

[24] 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.

[25] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker. Bandwidth-efficient threshold EC-DSA revisited: Online/offline extensions, identifiable aborts proactive and adaptive security. *Theor. Comput. Sci.*, 939:78–104, 2023. doi: 10.1016/J.TCS.2022.10.016. URL https://doi.org/10.1016/j.tcs.2022.10.016.

[26] R. Cohen, I. Haitner, N. Makriyannis, M. Orland, and A. Samorodnitsky. On the round complexity of randomized byzantine agreement. *J. Cryptol.*, 35(2):10, 2022. doi: 10.1007/S00145-022-09421-7. URL https://doi.org/10.1007/s00145-022-09421-7.

[27] R. Cramer and V. Shoup. Signature schemes based on the strong RSA assumption. *ACM Trans. Inf. Syst. Secur.*, 3(3):161–185, 2000. doi: 10.1145/357830.357847. URL https://doi.org/10.1145/357830.357847.

[28] A. P. K. Dalskov, M. Keller, C. Orlandi, K. Shrishak, and H. Shulman. Securing DNSSEC keys via threshold ECDSA from generic MPC. *IACR Cryptology ePrint Archive*, 2019:889, 2019.

[29] 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.

[30] I. Damgård, T. P. Jakobsen, J. B. Nielsen, J. I. Pagter, and M. B. Østergård. Fast threshold ecdsa with honest majority. Cryptology ePrint Archive, Report 2020/501, 2020. https://eprint.iacr.org/2020/501.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] J. Doerner, Y. Kondi, E. Lee, and a. shelat. Threshold ecdsa in three rounds. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 178–178, Los Alamitos, CA, USA, may 2024. IEEE Computer Society. doi: 10.1109/SP54263.2024.00178. URL https://doi.ieeecomputersociety.org/10.1109/SP54263.2024.00178.

[36] O. Friedman, A. Marmor, D. Mutzari, O. Sadika, Y. C. Scaly, Y. Spiizer, and A. Yanai. 2PC-MPC: Emulating two party ECDSA in large-scale MPC. Cryptology ePrint Archive, Paper 2024/253, 2024.

[37] 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.

[38] 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.

[39] R. Gennaro and S. Goldfeder. One round threshold ecdsa with identifiable abort. Cryptology ePrint Archive, Report 2020/540, 2020. https://eprint.iacr.org/2020/540.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] A. Gągol and D. Straszak. Threshold ecdsa for decentralized asset custody. Cryptology ePrint Archive, Report 2020/498, 2020. https://eprint.iacr.org/2020/498.

[45] I. Haitner, Y. Lindell, A. Nof, and S. Ranellucci. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. Cryptology ePrint Archive, Paper 2018/987, 2018.

[46] 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.

[47] 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.

[48] Y. Kondi, B. Magri, C. Orlandi, and O. Shlomovits. Refresh when you wake up: Proactive threshold wallets with offline devices. *IACR Cryptology ePrint Archive*, 2019:1328, 2019. URL https://eprint.iacr.org/2019/1328.

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

[50] 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.

[51] Y. Lindell and A. Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 1837–1854, 2018. doi: 10.1145/3243734.3243788. URL https://doi.org/10.1145/3243734.3243788.

[52] Y. Lindell, A. Nof, and S. Ranellucci. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody (2018 version). *IACR Cryptology ePrint Archive*, 2018:987, 2018. URL https://eprint.iacr.org/archive/2018/987/20181018:122733.

[53] 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.

[54] N. Makriyannis, O. Yomtov, and A. Galansky. Practical key-extraction attacks in leading MPC wallets. Cryptology ePrint Archive, Paper 2023/1234, 2023.

[55] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, Dec 2008. URL https://bitcoin.org/bitcoin.pdf. Accessed: 2015-07-01.

[56] National Institute of Standards and Technology. Digital signature standard (dss). Federal Information Processing Publication 186-5 (Draft), 2019. URL https://doi.org/10.6028/NIST.FIPS.186-5-draft.

[57] 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.

[58] 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.

[59] 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.

[60] 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.

[61] V. Shoup and J. Groth. On the security of ecdsa with additive key derivation and presignatures. In *EUROCRYPT*. Springer-Verlag, 2022.

[62] 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.

[63] H. Xue, M. H. Au, M. Liu, K. Y. Chan, H. Cui, X. Xie, T. H. Yuen, and C. Zhang. Efficient multiplicative-to-additive function from joye-libert cryptosystem and its application to threshold ecdsa. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 2974–2988. ACM, 2023. doi: 10.1145/3576915.3616595.

A Missing Sigma Protocols

A.1 Discrete Logarithm Proofs

Figure 22 is a Σ-protocol for the relation $\boldsymbol{R}_{\mathsf{sch}}$.

FIGURE 22 (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 $g^z = A \cdot X^e \in \mathbb{G}$.

Figure 22: Schnorr PoK—Π^{sch}

Figure 23 is a Σ-protocol for the relation $\boldsymbol{R}_{\mathsf{elog}}$.

FIGURE 23 (Dlog with El-Gamal Commitment—Π^{elog})

- **Inputs:** Common input is $(\mathbb{G}, g, L, M, X, Y, h)$.
  The Prover has secret input (y, λ) such that $L = g^\lambda$, $M = g^y X^\lambda$ and $Y = h^y$.

1. Prover samples
$$\alpha, m \leftarrow \mathbb{F}_q \text{ and computes } \begin{cases} A = g^\alpha, N = g^m X^\alpha \\ B = h^m \end{cases}$$
   and sends (A, N, B) to the verifier.
2. Verifier replies with $e \leftarrow \pm q$.
3. Prover Prover sends (z, u) to the verifier where $z = \alpha + e\lambda \mod q$ and $u = m + ey \mod q$.

- **Equality Checks**: $g^z = A \cdot L^e$ and $g^u X^z = N \cdot M^e$ and $h^u = B \cdot Y^e \in \mathbb{G}$.

Figure 23: Dlog with El-Gamal Commitment—Π^{elog}

A.1.1 Completeness, HVZK & Soundness of Π^{elog}

Completeness & HVZK. The completeness and HVZK properties hold perfectly. Namely, an honest prover generates a valid proof with probability 1, and, for the distribution $(z, u) \leftarrow \mathbb{F}_q^2$, $e \leftarrow \pm q$ and $(A, N, B) = (g^z \cdot L^{-e}, g^u X^z \cdot M^{-e}, h^u \cdot Y^{-e})$ yields a transcript (A, N, B, e, z, u) that is identically distributed with an honestly-generated proof. □

Special Soundness & Soundness. It is immediate that any two accepting transcripts of the form (A, N, B, e, z, u) and (A, N, B, e', z', u') for $e \neq e'$ yield that $(\mathbb{G}, g, L, M, X, Y, h) \in \boldsymbol{R}_{\mathsf{elog}}$. Consequently, since special soundness holds unconditionally, if $(\mathbb{G}, g, L, M, X, Y, h) \notin \boldsymbol{R}_{\mathsf{elog}}$ then the soundness property holds with probability $|\pm q|/|\pm q \times \pm q| = 1/q$ over the choice of e. □

A.2 Range Proof w/ El-Gamal Commitment ($\Pi^{\mathsf{enc\text{-}elg}}$)

Figure 24 is a Σ-protocol for the relation $\boldsymbol{R}_{\mathsf{enc\text{-}elg}}$.

FIGURE 24 (Range Proof w/ EL-Gamal Commitment—$\Pi^{\mathsf{enc\text{-}elg}}$)

- **Setup:** Auxiliary safe bi-prime $\hat{N}$ and Pedersen parameters $s, t \in \mathbb{Z}^*_{\hat{N}}$.
- **Inputs:** Common input is $(\mathbb{G}, q, g, N_0, C, A, B, X)$.
  The Prover has secret input (x, ρ, a, b) such that $x \in \pm 2^{\ell}$, and $C = (1+N_0)^x \cdot \rho^{N_0} \mod N_0^2$ and $(A, B, X) = (g^a, g^b, g^{ab+x}) \in \mathbb{G}^3$.

1. Prover samples

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

   and sends (S, T, D, Y, Z) 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

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

- **Equality Checks**:

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

- **Range Check**:

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

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

Figure 24: Range Proof w/ EL-Gamal Commitment—$\Pi^{\mathsf{enc\text{-}elg}}$

A.3 Paillier Operation with Group Commitment in Range ZK ($\Pi^{\mathsf{aff\text{-}g}}$)

In Figure 25 we give a Σ-protocol for tuples of the form $(\boldsymbol{I} = \pm 2^{\ell}, \boldsymbol{J} = \pm 2^{\ell'}, C, Y, X; x, y, k, r_0)$ satisfying relation $\boldsymbol{R}_{\mathsf{aff\text{-}g}}$. 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).

FIGURE 25 (Paillier Affine Operation with Group Commitment in Range ZK—$\Pi^{\mathsf{aff\text{-}g}}$)

- **Setup:** Auxiliary Paillier Modulus $\hat{N}$ and 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

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

   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**:

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

- **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 25: Paillier Affine Operation with Group Commitment in Range ZK—$\Pi^{\mathsf{aff\text{-}g}}$

A.4 Small-Factor Proof (Π^{fac})

Figure 26 is a Σ-protocol for the relation $\boldsymbol{R}_{\mathsf{fac}}$.

FIGURE 26 (Small-Factor Proof)

- **Setup:** Auxiliary safe bi-prime $\hat{N}$ and Pedersen parameters $s, t \in \mathbb{Z}^*_{\hat{N}}$.
- **Inputs:** Common input RSA modulus $N_0 > 2^{4\ell}$. The Prover has secret input (p, q) such that $p, q < \pm\sqrt{N_0} \cdot 2^{\ell}$.

1. Prover samples

$$\alpha, \beta \leftarrow \pm 2^{\ell+\varepsilon} \cdot \sqrt{N_0} \text{ and } \begin{cases} \mu, \nu \leftarrow \pm 2^{\ell} \cdot \hat{N} \\ r \leftarrow \pm 2^{\ell+\varepsilon} \cdot N_0 \cdot \hat{N} \\ x, y \leftarrow \pm 2^{\ell+\varepsilon} \cdot \hat{N} \end{cases}, \text{ and computes } \begin{cases} P = s^p t^\mu,\ Q = s^q t^\nu \mod \hat{N} \in \mathbb{Z}^*_{\hat{N}} \\ A = s^\alpha t^x \mod \hat{N} \in \mathbb{Z}^*_{\hat{N}} \\ B = s^\beta t^y \mod \hat{N} \in \mathbb{Z}^*_{\hat{N}} \\ T = Q^\alpha t^r \mod \hat{N} \in \mathbb{Z}^*_{\hat{N}} \end{cases},$$

   and sends (P, Q, A, B, T) to the verifier.
2. Verifier replies with $e \leftarrow \pm 2^{\ell}$
3. Prover sends (z_1, z_2, w_1, w_2, v) to the verifier, where

$$\begin{cases} z_1 &= \alpha + ep \\ z_2 &= \beta + eq \\ w_1 &= x + e\mu \\ w_2 &= y + e\nu \\ v &= r - e \cdot \nu p \end{cases}.$$

- **Equality Checks**: Verifier set $R = s^{N_0}$ and checks

$$\begin{cases} s^{z_1} t^{w_1} = A \cdot P^e \mod \hat{N} \in \mathbb{Z}^*_{\hat{N}} \\ s^{z_2} t^{w_2} = B \cdot Q^e \mod \hat{N} \in \mathbb{Z}^*_{\hat{N}} \\ Q^{z_1} t^{v} = T \cdot R^e \mod \hat{N} \in \mathbb{Z}^*_{\hat{N}} \end{cases}$$

- **Range Check**:

$$z_1, z_2 \in \pm\sqrt{N_0} \cdot 2^{\ell+\varepsilon}$$

  The proof guarantees that each $p, q > 2^{\ell}$ (assuming $2^{2\ell+\varepsilon} \approx \sqrt{N_0}$).

Figure 26: Small-Factor Proof

Special Soundness. Analogously to all other range proofs, the extractor can get p, q, μ, ν to decommit P & Q and $Q^p = s^{pq} t^{\nu p} = s^{N_0} t^{\nu p}$. Notice that if $N_0 \neq pq$, then the binding property of Pedersen is broken which, in turn, breaks strong RSA. □

HVZK. Sample $P, Q \leftarrow \langle t \rangle$. Sample $z_1, z_2 \leftarrow \pm 2^{\ell+\varepsilon} \cdot \sqrt{N_0}$ and $w_1, w_2 \leftarrow \pm 2^{\ell+\varepsilon} \cdot \hat{N}$ and $v \leftarrow 2^{\ell+\varepsilon} \cdot N_0 \cdot \hat{N}$ and $\sigma \leftarrow 2^{\ell} \cdot N_0 \cdot \hat{N}$. Finally, sample e and set A, B and T accordingly. □

A.5 Setup-less Affine Operation w/ Group Commitment ($\Pi^{\mathsf{aff\text{-}g*}}$)

Figure 27 is a Σ-protocol for the relation $\boldsymbol{R}_{\mathsf{aff\text{-}g}}$.

FIGURE 27 (Paillier Affine Operation with Group Commitment in Range ZK—$\Pi^{\mathsf{aff\text{-}g*}}$)

- **Inputs:** Common input is $(\mathbb{G}, g, N_0, N_1, C, D, Y, X)$ and security parameter κ. The Prover has secret input (x, y, ρ, μ) such that $x \in \pm 2^{\ell}$, $y \in \pm 2^{\ell'}$, $g^x = X$, $(1+N_1)^y \mu^{N_1} = Y \mod N_1^2$, and $D = C^x (1+N_0)^y \cdot \rho^{N_0} \mod N_0^2$.

1. For $j \in [\kappa]$, Prover samples $\alpha_j \leftarrow \pm 2^{\ell+\varepsilon}$ and $\beta_j \leftarrow \pm 2^{\ell'+\varepsilon}$ and

$$\begin{cases} r_j \leftarrow \mathbb{Z}^*_{N_0} \\ s_j \leftarrow \mathbb{Z}^*_{N_1} \end{cases} \quad \text{and computes} \quad \begin{cases} A_j = C^{\alpha_j} \cdot ((1+N_0)^{\beta_j} \cdot r_j^{N_0}) \mod N_0^2 \in \mathbb{Z}^*_{N_0^2} \\ R_j = g^{\alpha_j} \in \mathbb{G} \\ B_j = (1+N_1)^{\beta_j} s_j^{N_1} \mod N_1^2 \in \mathbb{Z}^*_{N_1^2} \end{cases}$$

   and sends $(A_j, B_j, R_j)_{j=1}^{\kappa}$ to the Verifier.
2. Verifier replies with $\{e_j \leftarrow \{0,1\}\}_{j=1}^{\kappa}$.
3. Prover Prover sends $(z_j, z'_j, w_j, \lambda_j)_{j=1}^{\kappa}$ to the Verifier where

$$\begin{cases} z_j = \alpha_j + e_j \cdot x \\ z'_j = \beta_j + e_j \cdot y \\ w_j = r_j \cdot \rho^{e_j} \mod N_0 \\ \lambda_j = s_j \cdot \mu^{e_j} \mod N_1 \end{cases}$$

- **Equality Checks**:

$$\begin{cases} C^{z_j} (1+N_0)^{z'_j} w_j^{N_0} = A_j \cdot D^{e_j} \mod N_0^2 \in \mathbb{Z}^*_{N_0^2} \\ g^{z_j} = R_j \cdot X^{e_j} \in \mathbb{G} \\ (1+N_1)^{z'_j} \lambda_j^{N_1} = B_j \cdot Y^{e_j} \mod N_1^2 \in \mathbb{Z}^*_{N_1^2} \end{cases}$$

- **Range Check**:

$$\begin{cases} z_j \in \pm 2^{\ell+\varepsilon} \\ z'_j \in \pm 2^{\ell'+\varepsilon} \end{cases}$$

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

Figure 27: Paillier Affine Operation with Group Commitment in Range ZK—$\Pi^{\mathsf{aff\text{-}g*}}$

A.6 Paillier Special Decryption in the Exponent (Π^{dec})

Figure 27 is a Σ-protocol for the relation R_{dec} from Section 3.2.2.

FIGURE 28 (Paillier Special Decryption in the Exponent—Π^{dec})

- **Inputs:** Common input is $(\mathbb{G}, g, q, N_0, \boldsymbol{I}, \boldsymbol{J}, K, X, D, S)$ and security parameter κ. The Prover has secret input x, y and ρ such that $(y, x) \in \boldsymbol{J} \times \boldsymbol{I}$ and $\rho \in \mathbb{Z}^2_{N_0}$ such that $(1+N_0)^z \cdot \rho^{N_0} = K^x \cdot D \mod N_0^2$ and $(S, X) = (g^y, g^x)$.

1. Prover samples $\{\alpha_j \leftarrow \boldsymbol{I}_\varepsilon, \beta_j \leftarrow \boldsymbol{J}_\varepsilon\}_{j=1}^{\kappa}$ and $\{r_j \leftarrow \mathbb{Z}^*_{N_0}\}_{j=1}^{\kappa}$ and calculates for all $j \in [\kappa]$:
$$\begin{cases} A_j = K^{-\alpha_j} \cdot (1+N_0)^{\beta_j} \cdot r_j^{N_0} \mod N_0^2 \in \mathbb{Z}^*_{N_0^2} \\ (B_j, C_j) = (g^{\beta_j}, g^{\alpha_j}) \in \mathbb{G}^2 \end{cases}$$
and sends $(A_j, B_j, C_j)_{j=1}^{\kappa}$ to the verifier.
2. Verifier replies with $\{e_j \leftarrow \{0,1\}\}_{j=1}^{\kappa}$
3. Prover sends $(z_j, w_j, \nu_j)_{j=1}^{\kappa}$ where
$$\begin{cases} z_j &= \alpha_j + e_j \cdot x \\ w_j &= \beta_j + e_j \cdot y \\ \nu_j &= r_j \cdot \rho^{e_j} \mod N_0 \end{cases}$$

- **Equality Checks**: For all $j \in [\kappa]$, verify
$$\begin{cases} (1+N_0)^{w_j} \cdot \nu_j^{N_0} \cdot K^{-z_j} = A \cdot D^{e_j} \mod N_0^2 \in \mathbb{Z}^*_{N_0^2} \\ (g^{z_j}, g^{w_j}) = (C_j \cdot X^{e_j}, B_j \cdot S^{e_j}) \in \mathbb{G}^2 \end{cases}$$

- **Range Check**: For all $j \in [\kappa]$, verify
$$(z_j, w_j) \in (\boldsymbol{I}_\varepsilon, \boldsymbol{J}_\varepsilon)$$

Figure 28: Paillier Special Decryption in the Exponent—Π^{dec}

B Proof of Theorem 6.7

Fix $\mathcal{Z}$ and let $\boldsymbol{\tau}_\Sigma$ denote the transcript (Notation 6.2) of Σ viewed as a random variable, and let α denote the probability that $\boldsymbol{\tau}_\Sigma$ contains a fault misattribution (Definition 6.4). Let d and ℓ denote a bound on the number of RO queries and the bit length of the output space of the RO. Let ν denote the smallest statistical soundness parameter in the protocol.

Proposition B.1. *Using the notation above, there exists efficient PPTM $\mathcal{A}$ such that:*

$$\Pr_{(N,C)\leftarrow\mathsf{sRSA}(1^\kappa)}[(m,e)\leftarrow\mathcal{A}(1^\kappa,N,C)\ \ s.t.\ \ m^e=C \mod N \wedge e\notin\{1,-1\}]\geq(\alpha-d\nu)^2/9d-2^{-\ell}$$

B.1 Proof of Proposition B.1

Recall the relation $\boldsymbol{R}_{\mathsf{elog}}, \boldsymbol{R}_{\mathsf{enc\text{-}elg}}, \boldsymbol{R}_{\mathsf{mod}}, \boldsymbol{R}_{\mathsf{prm}}, \boldsymbol{R}_{\mathsf{fac}}, \boldsymbol{R}_{\mathsf{dec}}$ from Section 3.2. Define $\mathcal{R}_{\mathrm{D}}$ interacting with $\mathcal{Z}$ in the in-between, hybrid experiment D such that $\mathcal{R}_{\mathrm{D}}$ runs the code of the honest parties in Σ except that it outputs fail and halts in any one of the cases below. We note that $\mathcal{R}_{\mathrm{D}}$ is *not* an efficient machine.

1. $N_i\notin\boldsymbol{R}_{\mathsf{mod}}$ for some $i\in\boldsymbol{P}$ and the associated proof verifies according to Π^{mod}.
2. $(\Gamma_i,g,B_{i,1},B_{i,2},Y_i)\notin\boldsymbol{R}_{\mathsf{elog}}$ for some $i\in\boldsymbol{P}$ and the associated proof verifies according to Π^{elog}.
3. $(\Delta_i,\Gamma,A_{i,1},A_{i,2},Y_i)\notin\boldsymbol{R}_{\mathsf{elog}}$ for some $i\in\boldsymbol{P}$ and the associated proof verifies according to Π^{elog}.
4. $(\boldsymbol{I}_\varepsilon,\boldsymbol{J}_\varepsilon,D_{j,i},K_i,F_{i,j},\Gamma_j)\notin\boldsymbol{R}_{\mathsf{aff\text{-}g}}$ for some $i,j\in\boldsymbol{P}$ and the associated proof verifies according to $\Pi^{\mathsf{aff\text{-}g}*}$.
5. $(\boldsymbol{I}_\varepsilon,\boldsymbol{J}_\varepsilon,\hat{D}_{j,i},K_i,\hat{F}_{i,j},X_j)\notin\boldsymbol{R}_{\mathsf{aff\text{-}g}}$ for some $i,j\in\boldsymbol{P}$ and the associated proof verifies according to $\Pi^{\mathsf{aff\text{-}g}*}$.
6. $(\boldsymbol{I}_\varepsilon,\boldsymbol{J}_\varepsilon,K_i,\Gamma_i,D_i,g^{\delta_i},g)\notin\boldsymbol{R}_{\mathsf{dec}}$ for some $i\in\boldsymbol{P}$ and the associated proof verifies according to Π^{dec}.
7. $(\boldsymbol{I}_\varepsilon,\boldsymbol{J}_\varepsilon,K_i,X_i,\hat{D}_i,S_i,\Gamma)\notin\boldsymbol{R}_{\mathsf{dec}}$ for some $i\in\boldsymbol{P}$ and the associated proof verifies according to Π^{dec}.

Claim B.2. *Define $\boldsymbol{\tau}_{\mathrm{D}}$ with respect to* D *analogously to $\boldsymbol{\tau}_\Sigma$. It holds that* $\mathrm{SD}(\boldsymbol{\tau}_\Sigma,\boldsymbol{\tau}_{\mathrm{D}})\leq d\cdot\nu$.

Proof. Corollary of the statistical soundness of each proof. □

Putting everything together. Proposition B.1 is a corollary of the forking lemma [4] (Lemma 7.16), Claims 5.1 and B.3 and Theorem 5.2. The security proof is essentially identical to the security proof of Proposition 7.12 in Section 7.1.7.

Claim B.3. *Let cheat_Σ denote the probability that $\boldsymbol{\tau}_\Sigma$ contains a malformed tuple for $\boldsymbol{R}_{\mathsf{fac}}$ or $\boldsymbol{R}_{\mathsf{enc\text{-}elg}}$. Then,*

$$\Pr_{\boldsymbol{\tau}_\Sigma}[\mathsf{cheat}_\Sigma]\geq\alpha-d\cdot\nu.$$

Proof. Similarly to the proof of Claim 7.14, we define $\mathcal{R}_{\mathrm{D}^*}$ analogously to $\mathcal{R}_{\mathrm{D}}$ for a new experiment D^*, where $\mathcal{R}_{\mathrm{D}^*}$ additionally checks that all the relevant tuples are well formed with respect to $\boldsymbol{R}_{\mathsf{fac}}$ or $\boldsymbol{R}_{\mathsf{enc\text{-}elg}}$. Define $\mathsf{cheat}_{\mathrm{X}}$ for $\mathrm{X}\in\{\mathrm{D},\mathrm{D}^*\}$ analogously to cheat_Σ. We deduce that $\Pr_{\boldsymbol{\tau}_{\mathrm{D}}}[\mathsf{cheat}_{\mathrm{D}}]=\Pr_{\boldsymbol{\tau}_{\mathrm{D}^*}}[\mathsf{cheat}_{\mathrm{D}^*}]$ using a similar argument to Claim 7.14. In conclusion, since $\boldsymbol{\tau}_{\mathrm{D}^*}$ does not contain fault misattributions when all the tuples are well-formed, and since D and Σ are statistically $(d\cdot\nu)$-indistinguishable, the claim follows. □

C Complexity Estimates

We provide computation and communication cost-analysis of our protocol's components in Table 1, mostly derived from the cost-analysis of each of our NIZKs, presented in Table 2.

Component	*Rounds*	*Computation*	*Communication*
Key Generation	3	$n \times (2\mathbf{G})$	$n \times (4\kappa)$
Aux Info. & Key Refresh	3	$n \times (2n\mathbf{G} + 2m \cdot \mathbf{N} + 2\mathbf{N}^2)$	$n \times (2n\kappa + (2m+5)\nu)$
Pre-Signing	3	$n \times (26\mathbf{G} + 48\mathbf{N} + 26\mathbf{N}^2)$	$n \times (65\kappa + 48\nu)$
Signing	1	0	$n \times (\kappa)$

Table 1: We report *total costs per party* (over all rounds); $\mathbf{G}, \mathbf{N}, \mathbf{N}^2$ denote computing exponentiation in the EC group $\mathbb{G}$ and rings $\mathbb{Z}_N, \mathbb{Z}_{N^2}$, respectively. For communication, we report *total incoming communication* (for each party over all rounds); κ and ν correspond to the message size of EC group element and Paillier plaintext ring elements, respectively. Constants and hash (random oracle) invocations were omitted from above.

ZK-Proof	*Computation (Prover)*	*Computation (Verifier)*	*Communication*
Π^{sch}	$1\mathbf{G}$	$2\mathbf{G}$	2κ
Π^{elog}	$4\mathbf{G}$	$7\mathbf{G}$	5κ
$\Pi^{\mathsf{enc\text{-}elg}}$	$3\mathbf{G} + 5\mathbf{N} + 1\mathbf{N}^2$	$5\mathbf{G} + 3\mathbf{N} + 2\mathbf{N}^2$	$9\kappa + 6\nu$
$\Pi^{\mathsf{aff\text{-}g}}$	$1\mathbf{G} + 10\mathbf{N} + 3\mathbf{N}^2$	$2\mathbf{G} + 6\mathbf{N} + 5\mathbf{N}^2$	$17\kappa + 12\nu$
Π^{mod}	$2m \cdot \mathbf{N}$	$m \cdot \mathbf{N}$	$2m \cdot \nu$
Π^{prm}	$m \cdot \mathbf{N}$	$m \cdot \mathbf{N}$	$2m \cdot \nu$

Table 2: In the above, m denotes the amplification parameter of Π^{mod} and Π^{prm}. In the remaining ZK-Range-Proofs, ℓ, ℓ', and ε are respectively 1, 5, and 2 times the bit-length of the EC element κ. (See Appendix C.1 for the rationale behind this choice of parameters.)

C.1 Concrete Security

We briefly discuss the choice of concrete parameters and their implications for 'concrete security.' Given the bounds below, it follows that each secret is masked by at least ℓ bits. Specifically, $(\ell + \varepsilon) - (\ell + \kappa) \geq \ell$ and $(\ell' + \varepsilon) - (\ell' + \kappa) \geq \ell$ in the ZK proofs $\Pi^{\mathsf{enc\text{-}elg}}$ and $\Pi^{\mathsf{aff\text{-}g}}$, where, for example, in $\Pi^{\mathsf{enc\text{-}elg}}$, α is chosen large enough to hide $e \cdot x$ for $e \in \pm q$ and $x \in \pm 2^{\ell}$. Additionally, $\ell' - (2\ell + \varepsilon) \geq \ell$ for the additive mask in the Paillier affine operation, where β and $\hat{\beta}$ are chosen large enough to hide γk and xk. Furthermore, the Paillier modulus bit length is chosen large enough so that the largest possible plaintext (of size $\ell' + \varepsilon$) does not overflow.

$$\begin{cases} \kappa = \log(q) \\ \ell \geq \kappa \\ \varepsilon \geq \ell + \kappa \\ \ell' \geq \ell + \varepsilon + 2\ell \\ \log(N) \geq \ell' + \varepsilon \end{cases}$$

By setting $(\ell, \varepsilon, \ell') = (\kappa, 2\kappa, 5\kappa)$ for $\kappa = 256$ (e.g., for secp256k1), the bit security is dictated by the length of the Paillier and Pedersen moduli, and the amplification parameter m in Π^{prm} and Π^{mod}. Using the NIST recommendations [3] for the RSA moduli, the overall bit security is 112 for $(\log(N), \log(\hat{N}), m) = (2048, 2048, 112)$ and 128 for $(\log(N), \log(\hat{N}), m) = (3072, 3072, 128)$. Aiming for higher bit security with secp256k1 is somewhat pointless as the bit security of the curve itself is 128.

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 efficiently computable* $y \in \mathbb{Z}_M^*$ *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* $N = pq$ *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.5. *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.6. *Let* N *be the product of exactly two arbitrary primes* p *and* q. *Let* $\boldsymbol{a} \leftarrow \mathbb{Z}_{\ell \cdot N}$ *and* $\boldsymbol{b} \leftarrow \mathbb{Z}_{\varphi(N)}$. *It holds that* $\mathrm{SD}(\boldsymbol{a} \mod \varphi(N), \boldsymbol{b}) \leq \frac{1}{\ell}$.

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

E On Enhanced Existential Unforgeability

We use the generic model (GGM) to provide some support to the enhanced unforgeability assumption. Namely, in the GGM, the group operations are handled via an oracle and each group element is encoded using a unique random identifier (so there is no way to relate the discrete log to the group element via the identifier). Thus, if we can show that that a property holds in the GGM, then it follows that any adversary that breaks this property somehow crucially uses the encoding of the group elements (e.g. the underlying curve) in their attack, or, in other words, no attacker that uses the group generically can break the property in question.

In the paper, we show that any generic attacker that manages to forge signatures in the enhanced ECDSA game can be transformed into an attacker that finds non-trivial relations in the hash function. Assuming it is infeasible to find such relation, e.g. by modeling the hash as an RO, it follows that no generic attacker can break the enhanced unforgeability game. Hereafter, $\mathcal{H}$ denotes the ECDSA hash function, modelled as a random oracle, i.e. $\mathcal{F} = \mathcal{H}$.

E.1 Forgeries with Arbitrary Leakage in the GGM

Let $\boldsymbol{G}$ denote the labels of the elements in the elliptic curve $(\mathbb{G}, g, q)$. As $\mathbb{G}$ denotes an elliptic curve, every $(A, A^{-1}) \in \mathbb{G}^2$ has the form $((\alpha, 0), (\alpha, 1))$ or $((\alpha, 1), (\alpha, 0)) \in \boldsymbol{G}^2$. Define function $\tau : \boldsymbol{G} \to \mathbb{F}_q$ such that $\tau(H) = \eta \in \mathbb{F}_q$ for $H \in \boldsymbol{G}$ of the form $(\eta, 0)$ or $(\eta, 1)$. Notice that this map is efficiently invertible $\tau^{-1} : \mathbb{F}_q \to \{\{G, H\} \text{ s.t. } G, H \in \boldsymbol{G}\} \cup \{\perp\}$ such that

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

Brief overview of the EC Generic Group Model. The generic group is defined via a random bijective map $\mu : \mathbb{G} \to \boldsymbol{G}$ that preserves τ and a group-oracle $\mathcal{O} : \boldsymbol{G} \times \boldsymbol{G} \to \boldsymbol{G}$ such that $\mu(gh) = \mathcal{O}(\mu(g), \mu(h))$ and $\tau(\mu(g)) = \tau(\mu(g^{-1}))$, 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 $G * H = \mathcal{O}(G, H)$.

FIGURE 29 (ECDSA Experiment in Generic Group w/ Enhanced Signing Oracle)

- Group Oracle $\mathcal{O}$:
  - On input (X, Y), return $Z = X * Y$.
- Signing oracle $\mathcal{S}^{\mathcal{O}}$:
  - On input `pub-key`, 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 $m = \mathcal{H}(\text{msg})$.
    2. Remove R from $\boldsymbol{R}$ and add (R, msg, σ) to $\boldsymbol{S}$.
- Group 'accounting':

  A list $\boldsymbol{Q} = \{\ell_Z\}_Z$ is maintained for every queried Z under the following rule set.

  1. $\ell_Z = (\alpha_A)_{A \in \boldsymbol{R} \cup \{G,H\}}$ such that $Z = G^{\alpha_G} H^{\alpha_H} \prod_{R \in \boldsymbol{R}} R^{\alpha_R}$
  2. If Z was obtained as $X * Y$ in a group oracle invocation then $\ell_Z = \ell_X + \ell_Y \mod q$
  3. If $Z \in \boldsymbol{R} \cup \{G, H\}$, then $\ell_Z = (0, \ldots, 0, 1, 0, \ldots, 0)$, where the 1-value is indexed by Z.
  4. For every signature σ obtained for m and $R = G^{m/\sigma} H^{\tau(R)/\sigma}$, for every $\ell_Z \in \boldsymbol{Q}$ do:
     (a) Remove the R-th entry of ℓ_Z and let c_Z denote the removed value.
     (b) Update $\ell_Z := \ell_Z + c_Z \cdot (m/\sigma, \tau(R)/\sigma, 0, \ldots, 0) \mod q$.

Figure 29: ECDSA Experiment in Generic Group w/ Enhanced Signing Oracle

Theorem E.1. *Let $\mathcal{A}$ be an algorithm in the generic group experiment with enhanced signing oracle making t queries to the group and signing oracle. Assume that $\mathcal{A}$ outputs a forgery with probability α. Then, there exists $\mathcal{B}$ with blackbox access to $\mathcal{A}$ such that*

$$\Pr_{e \leftarrow \mathbb{F}_q}[(x, y) \leftarrow \mathcal{B}(e) \;\; s.t. \;\; \mathcal{H}(x)/\mathcal{H}(y) = e] \geq \alpha/2t^2 - \mathsf{poly}(1/q).$$

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

Claim E.2. *Using the notation from Theorem E.1, there exists $\mathcal{B}$ with blackbox access to $\mathcal{A}$ such that*

$$\Pr_{e_1, \ldots, e_t \leftarrow \mathbb{F}_q}[(x, y) \leftarrow \mathcal{B}(e_1, \ldots, e_t) \;:\; \exists i \;\; s.t. \;\; \mathcal{H}(x)/\mathcal{H}(y) = e_i] \geq \alpha/2t - \mathsf{poly}(1/q).$$

E.1.1 Proof of Claim E.2

Remark E.3 (Simplifying Assumption). The $\mathsf{poly}(1/q)$ term comes from the event that the attacker $\mathcal{A}$ distinguishes between the 'real' generic group in Figure 29 and the simulated group presented further below. We ignore this term in the analysis. Furthermore, the term $1/2$ comes from the probability that $\tau^{-1}(\alpha) = \perp$ for a random α. We also ignore this term in the analysis. Furthermore, it is assumed that the attacker never queries the group and signing oracle on elements $Z \in \boldsymbol{G}$ that were never produced before. This assumption does not incur any loss of generality and simply makes the description of the experiments less tedious.

Using the notation from Figure 29, let ϕ denote the (efficient) function that maps group-elements to their representation with respect to $\boldsymbol{Q}$. Namely $\phi(Z) = \ell_Z$ as determined by $\boldsymbol{Q}$ in Figure 29. Let $\mathcal{A}$ denote an adversary that produces a forgery in Figure 29 and consider the reduction from Figure 30. In the end of the experiment, assume $\mathcal{A}$ outputs $x \in \{0,1\}^*$ and (F, ψ) such that $F = G^{\mathcal{H}(x)/\psi} * H^{\tau(F)/\psi}$ and $(\ldots, x, \ldots) \notin \boldsymbol{S}$.

FIGURE 30 (Reduction in Generic Group w/ Enhanced Signing Oracle)

- Input: $e_1, \ldots, e_t \in \mathbb{Z}_q$ and $\lambda \in \mathbb{N}$ and PPTM $\mathcal{A}$.
  - Initialize: ctr $= 1$.
- Group operation simulation:
  - On input (X, Y) from $\mathcal{A}$ to the group-oracle, do:
    1. If $\phi(Z) = \phi(X * Y)$ for some previously queried Z.
    2. Else If $\phi(X * Y) = (\alpha, \beta, 0, \ldots, 0)$ for $\alpha, \beta \neq 0$ do:
       (a) Sample $y \leftarrow \mathbb{F}_q$ and set $w = e_{\text{ctr}} \cdot \mathcal{H}(y)$ and $Z \leftarrow \tau^{-1}(\alpha^{-1} w \beta)$.
           If $\tau^{-1}(\alpha^{-1} w \beta) = \perp$, repeat the above step. Otherwise, increment ctr and carry on.
       (b) Return Z.
    3. Else if $\phi(X * Y) = (\alpha, \beta, \gamma_1, \ldots, \gamma_\ell)$ for some $\gamma_i \neq 0$ do:
       (a) Choose $j \leftarrow [\ell]$ and set $Z \leftarrow \tau^{-1}(e_{\text{ctr}} \cdot r_j)$, for $r_j = \tau(R_j)$.
           If $\tau^{-1}(e_{\text{ctr}} \cdot r_j) = \perp$, repeat the above step. Otherwise, increment ctr and carry on.
       (b) Return Z.

    Update $\boldsymbol{Q}$ according to the rule set from Figure 29
- Signing operation simulation:
  - On input `pub-key` from $\mathcal{A}$ to the signing-oracle, return $(G, H) \leftarrow \boldsymbol{G}^2$.
  - On input `pnt-request` from $\mathcal{A}$ to the signing-oracle, return $R \leftarrow \boldsymbol{G}$, and add R to $\boldsymbol{R}$.
  - On input $(\texttt{sign}, \text{msg}, R)$ from $\mathcal{A}$ to the signing-oracle, if $R \in \boldsymbol{R}$ set $m = \mathcal{H}(\text{msg})$ and $r = \tau(R)$, and do:
    1. Choose $Z \leftarrow \boldsymbol{Q}$ such that $\phi(Z) = (\alpha, \beta, 0, \ldots, 0, \gamma, 0, \ldots, 0)$ for $\gamma \neq 0$ and $\beta m - r\alpha \neq 0$, and R denotes the index of γ.
       (a) Sample y and set $w = e_{\text{ctr}} \cdot \mathcal{H}(y)$ and $\sigma = \gamma(wr\zeta^{-1} - m) \cdot (\alpha - w\zeta^{-1}\beta)^{-1}$, for $\zeta = \tau(Z)$.
           Increment ctr.
       (b) If no such Z exists set $\sigma \leftarrow \mathbb{F}_q$.
    2. Return σ and remove R from $\boldsymbol{R}$, and add (R, msg, σ) to $\boldsymbol{S}$.

  Update $\boldsymbol{Q}$ according to the rule set from Figure 29

Figure 30: Reduction in Generic Group w/ Enhanced Signing Oracle

We begin by showing that in the following cases (depending on F), the transcript of the reduction yields a 'collision' (x, y) of the form $\mathcal{H}(x)/\mathcal{H}(y) = e$ for $e \leftarrow \mathbb{F}_q$.

Case 1. F was returned to the adversary in Item 2 of the group operation simulation. Using the notation from Figure 30, it holds that $F = G^\alpha H^\beta = G^{\mathcal{H}(x)/\psi} H^{f/\psi}$ for $f = \alpha^{-1}\mathcal{H}(y)e\beta$. Consequently $f/\psi = \beta$ and $\mathcal{H}(x)/\psi = \alpha$ and we deduce that $\mathcal{H}(x)/\mathcal{H}(y) = (\alpha\psi) \cdot (f^{-1}\alpha^{-1}e\beta) = e \cdot (\beta\psi f^{-1}) = e$.

Case 2. F was returned to the adversary in Item 3 of the group operation simulation, and, for $(R, y, \sigma) \in \boldsymbol{S}$, it holds that $F = G^\alpha H^\beta R^\gamma$ and $\alpha r - \beta m = 0$. Using the notation from Figure 30, it holds that $F = G^\alpha H^\beta R^\gamma$ for $\tau(F) = e \cdot r$ and $r = \tau(R)$. Next, retrieve (R, y, σ) from $\boldsymbol{S}$ such that $R = G^{\mathcal{H}(y)/\sigma} H^{\tau(R)/\sigma}$ and deduce that $F = G^{\alpha + \gamma m/\sigma} H^{\beta + \gamma r/\sigma}$. Deduce that $\beta/r + \gamma/\sigma = f/\psi = \frac{er}{\psi}$, and, since $\alpha = m\beta/r$, observe that $m(\beta/r + \gamma/\sigma) = \chi/\psi$ which implies that $e = \chi/m = \mathcal{H}(x)/\mathcal{H}(y)$.

Case 3. F was chosen by the simulator in Item 1 of the signing operation simulation. It holds that $F = G^{\alpha}H^{\beta}R^{\gamma} = G^{\alpha+\gamma m/\sigma}H^{\beta+\gamma r/\sigma} = G^{\mathcal{H}(x)/\psi}H^{f/\psi}$ for σ set as prescribed in Item 1 of the signing operation simulation i.e.

$$\sigma = \gamma(wrf^{-1} - m) \cdot (\alpha - wf^{-1}\beta)^{-1}$$

Therefore, letting $\chi = \mathcal{H}(x)$, since $\chi/\psi = \alpha + \gamma m/\sigma$ and $f/\psi = \beta + \gamma r/\sigma$, it follows that

$$\begin{aligned}
\chi(\beta + \gamma r/\sigma) &= f(\alpha + \gamma m/\sigma) \quad \Leftrightarrow \\
f\alpha - \chi\beta &= \frac{\gamma}{\sigma} \cdot (\chi r - mf) \quad \Leftrightarrow \\
f\alpha - \chi\beta &= \frac{\gamma}{\sigma} \cdot (\chi r - mf) \quad \Leftrightarrow \\
(f\alpha - \chi\beta)(\chi r - mf)^{-1} &= (wrf^{-1} - m)^{-1}(\alpha - wf^{-1}\beta) \quad \Leftrightarrow \\
(w - \chi)(r\alpha - \beta m) &= 0
\end{aligned}$$

which implies $\mathcal{H}(x)/\mathcal{H}(y) = e$ for $w = \mathcal{H}(y)e$ since $r\alpha - \beta m \neq 0$.

Simulation Indistinguishability. Next, we show that if the adversary forges in Figure 29, then the reduction from Figure 30 yields a forgery as well. It is sufficient to argue that the (simulated) group elements and signatures have the same distribution as in Figure 29. Hereafter, we refer to Figure 29 as the real experiment and to Figure 30 as the simulated experiment.

In the simulated experiment, all the points are simply random uniform elements in $\boldsymbol{G}$ (since $R \leftarrow \tau^{-1}(r)$ for $r \leftarrow \mathbb{F}_q$). The nontrivial part is to show that the signatures, i.e. the sigmas, are well-distributed. In the real experiment, $\sigma = k(\mathcal{H}(x) + rx)$ for a random k, so the signatures are simply uniform elements in $\mathbb{F}_q$ (since in the real experiment, elements k and $r = \tau(R)$ are independent). We show that the sigmas are (almost) uniform elements in the simulated experiment by showing that the map $w \mapsto \gamma(wr\zeta^{-1} - m) \cdot (\alpha - w\zeta^{-1}\beta)^{-1}$ is injective when $\gamma\zeta \neq 0$ and $r\alpha - m\beta \neq 0$. So, for $u, v \in \mathbb{F}_q$, if

$$\gamma(ur\zeta^{-1} - m) \cdot (\alpha - u\zeta^{-1}\beta)^{-1} = \gamma(vr\zeta^{-1} - m) \cdot (\alpha - v\zeta^{-1}\beta)^{-1} \quad \Leftrightarrow \tag{4}$$

$$(ur\zeta^{-1} - m) \cdot (\alpha - v\zeta^{-1}\beta) = (vr\zeta^{-1} - m) \cdot (\alpha - u\zeta^{-1}\beta) \quad \Leftrightarrow \tag{5}$$

$$ur\alpha + mv\beta = vr\alpha + mu\beta \quad \Leftrightarrow \tag{6}$$

$$(u - v)(r\alpha - m\beta) = 0 \tag{7}$$

Therefore, for random w, the simulated signature is uniform over $\mathbb{F}_q \setminus \{z\}$, where $z \in \mathbb{F}_q$ corresponds to the solitary field element not in the support of the function.

Putting everything together. We conclude by calculating the probability that the transcript contains a 'collision' of the form $\mathcal{H}(x)/\mathcal{H}(y) = e$. Let (F, x, ψ) denote $\mathcal{A}$'s forgery. If F was returned to $\mathcal{A}$ in Item 2 of the group operation simulation (case 1), then clearly the transcript yields a suitable collision. The other two cases are slightly more complicated because the reduction is required to guess the attempted forgery. For instance, in Item 3 of the group-operation simulation (case 2), the reduction picks a random (presigning) point R hoping that (F, x, ψ) together with $(R, \text{msg}, \sigma) \in \boldsymbol{S}$ will result in a 'collision'. Similarly, in Item 1 of the signing-operation simulation (case 3), the reduction chooses a random Z with the aim that $F = Z$.

In more detail, assume that F was first calculated of the form $G^{\alpha}H^{\beta}\prod_i R_i^{\gamma_i}$ and deduce the following about $F = G^{\alpha'}H^{\beta'}R^{\gamma'}$ where R denotes the last point to be signed on $m = \mathcal{H}(\text{msg})$ among $\{R_i\}_i$. If $\alpha' r + \beta' m = 0$, then R was chosen by the reduction in Item 3 of the group operation with probability at least $1/t$ (since t is an upper bound on the number of group-queries), or, if $\alpha' r + \beta' m \neq 0$, then F was chosen by the reduction in Item 1 of the signing operation with probability at least $1/t$ (since t is an upper bound on the number of presigning queries).

In summary, it holds that the transcript contains a suitable collision with probability α/t.

□

F Additional Comments

F.1 Extension to t-out-of-n

Our protocol can be extended to the t-out-of-n threshold case using Shamir secret sharing and appropriate modifications to the key-generation and key-refresh phases. The presigning phase remains additive t'-out-of-t' (for $t' \in \{t, \ldots, n\}$) as, using Lagrange coefficients, the parties locally convert their shares into additive shares in each presigning session involving t' parties. Consequently, the signing phase remains unchanged.

Regarding the security analysis, with suitable tweaks to the ideal functionality $\mathcal{F}_{\mathsf{tsig}}$, the analysis from this paper extends to the general threshold case as long as $n^{t_0} \in \mathsf{poly}(\kappa)$, for $t_0 = \min\{t, n-t\}$. Otherwise, the reduction from Section 8 does not run in polynomial time.

We propose an extension of our protocol to the t-out-of-n threshold case by discussing the items that need to be adapted for this case. Since the key-refresh phase implicitly addresses the key-generation phase, our focus will be on discussing the key-refresh phase.

F.1.1 Key-Refresh for the t-out-of-n case

$\mathcal{P}_i$ performs the following operation in the presence of all parties, as the protocol involves all n participants. Let $X_{i,0}$ denote the additive n-out-of-n public share of $\mathcal{P}_i$ at the end of a given epoch, obtained by applying the appropriate Lagrange coefficient,[21] and let $x_{i,0}$ denote the discrete log of $X_{i,0}$, i.e. $g^{x_{i,0}} = X_{i,0}$ and $\prod_{j=1}^{n} X_{j,n} = X$.

When obtaining the rid during the key-refresh, the parties are instructed to set $\mathsf{id}_j = \mathcal{H}(sid, rid, \mathcal{P}_j) \in \mathbb{F}_q$ as the Shamir identifier for each $\mathcal{P}_j$ in the new epoch. Then, each $\mathcal{P}_i$ samples $\vec{X}_i = (g^{x_{i,k}})_{k=1}^{t} \in \mathbb{G}^t$ as a vector of t group elements, where each $x_{i,k}$ represents the coefficient of the k-th monomial of a degree-$(t+1)$ polynomial with a constant term of $x_{i,0}$. All other values are sampled as the original protocol prescribes.

In the third round of the protocol, each $\mathcal{P}_i$ receives $C_{i,j} = z_{j,i} + \rho_{j,i} \mod q$ from $\mathcal{P}_j$, where $\rho_{j,i}$ is calculated as before and $z_{j,i} = \sum_{k=0}^{t} x_{j,k} \cdot \mathsf{id}_i^k \mod q$. This is verified by $\mathcal{P}_i$ using the formula $g^{z_{j,i}} = \prod_{k=0}^{t} X_{j,k}^{\mathsf{id}_i^k}$. All other checks are adapted from the previous protocol in the expected way (in particular there are t Schnorr proofs per party for each additional group element $X_{i,j}$).

In the end, if no error is detected, each $\mathcal{P}_i$ updates its secret share to $\sum_{j=1}^{n} z_{j,i} \mod q$. The public share of each counterparty ($\mathcal{P}_j$) is updated to $\prod_{k=0}^{n} \left(\prod_{\ell=1}^{n} X_{\ell,k}\right)^{\mathsf{id}_j^k}$. Additionally, the protocol updates the auxiliary information, including the Paillier and Pedersen tuples.

F.2 Comments on previous version of the key-refresh

The key-refresh protocol has been updated compared to the last iteration of this paper, specifically in how the parties calculate the ciphertexts $C_{i,j}$ in round 3 of the key-refresh phase. In the current version, $C_{i,j} = x_{j,i} + \rho_{j,i} \mod q$ (instead of a Paillier encryption), and the randomizers $\rho_{j,i}$ are calculated as the output of the RO on an input resulting from a DH tuple shared between $\mathcal{P}_i$ and $\mathcal{P}_j$.

In the previous version, each $C_{i,j}$ was a random Paillier encryption of $x_{j,i}$ under $\mathcal{P}_i$'s key. The reason for changing this approach, which was implicitly addressed in the description of the key-refresh phase in Section 4.2, is twofold:

1. First, DH-style encryption is much more efficient than Paillier encryption and better aligns with the protocol's intended design. The specific use of Paillier encryption for $C_{i,j}$ is not crucial; what matters is that these values are associated with freshly generated, preferably ephemeral, public keys.

2. Second, this change greatly simplifies the security analysis, as now adaptive security follows from static security under the DDH assumption (Section 8).

[21] Shamir secret shares allow the parties to generate additive shares for each set of privileged parties they belong to. Consequently, for each $\mathcal{P}_i$, there is a unique well-defined additive share $x_{i,0}$, derived from its Shamir share, corresponding to the set of privileged parties that includes all parties.

On the security of the previous version. As far as we know, the previous version of the key-refresh protocol is provably secure, though it introduces some challenges in the security proof. First, in the reduction to DCR, the simulator must guess whether the attacker will cheat to use a 'decryptable' Paillier key rather than the key resulting from the DCR-challenge distribution. Second, and more importantly, even if the attacker doesn't cheat, the simulator can no longer execute the protocol by simply running the code of the honest parties, as it cannot decrypt the ciphertexts of the honest parties. This means that proving adaptive security requires additional extraction steps, ultimately leading to reproving unforgeability from scratch.

To avoid this complication and to ensure that adaptive security follows neatly from the static case, we have opted for the new key-refresh protocol included in this paper.