Efficient Asymmetric Threshold ECDSA for MPC-based Cold Storage

Constantin Blokh*† Nikolaos Makriyannis† Udi Peled†

Abstract

Motivated by applications to cold-storage solutions for ECDSA-based cryptocurrencies, we present a new ECDSA protocol between n "online" parties and a single "offline" party, and the purpose of the protocol is to limit the exposure of the offline party to the outside world. Our protocol admits a *pre-signing* mode of operation for calculating preprocessed data for future signatures. The main novelty of our protocol is twofold.

- Pre-signing is lightweight and non-interactive; it consists of each party (offline or online) sending a single *independently-generated* short message per future signature, per online party (approx. 300B for typical choice of parameters).
- The signing phase is asymmetric in the following sense; to calculate the signature, it is enough for the offline party to receive a *single* short message from the outside world (approx. 300B in total).

We note that all previous ECDSA protocols (including the ones that support pre-signing) require many rounds of interaction at some stage of the execution between *all* parties. In contrast, our protocol minimizes the overall exposure of the offline party, and it is thus ideal for MPC-based cold storage. Regarding security, our protocol tolerates all-but-one *adaptive* corruptions, and it supports full proactive security. We prove security in the UC framework with a global random oracle under assumptions that are widely held in both the literature and in commercially-available MPC-based wallets; namely, strong RSA, DCR (Paillier assumption), DDH, and enhanced unforgeability of ECDSA.

On a technical level, first, building on existing protocols, we design a two-party protocol that we non-generically compile into a highly efficient $(n + 1)$-party protocol. Second, to achieve the aforementioned efficiency, we present a new batching technique for proving in zero-knowledge that the plaintexts of *practically any number* of Paillier ciphertexts all lie in a given range. The cost of the resulting *batch* proof is very close to that of the underlying non-batch proof, and the technique is applicable to arbitrary Sigma Protocols, i.e. three-round ZK protocols for one-way homomorphisms. Given the numerous applications of Sigma protocols, we view batching as a main contribution of independent interest.

---

*Authors are listed in alphabetical order.

†Fireblocks. Emails: costy@fireblocks.com, nikos@fireblocks.com, udi@fireblocks.com

Contents

1 Introduction

The *digital signature algorithm* (DSA) [28] in its elliptic curve variant (ECDSA) [37] is one of the most widely used signature schemes. ECDSA's popularity has surged in recent years because it is ubiquitous in the Blockchain space, where it is primarily used to sign transactions. For example, in Bitcoin, each transaction is accompanied by a datum, called a *signature*, generated using a secret key, such that all participants (miners, nodes, ...) may verify the validity of the transaction using the signature and publicly available data.

While ECDSA and digital signatures more broadly provide indispensable functionality and security (authenticity & integrity) to the various applications they make possible, most schemes suffer from the "single point of failure" problem, i.e. if the machine storing the secret key is compromised by a malicious agent, then this agent can impersonate the owner and sign any message/transaction on their behalf, e.g. causing a total loss of funds for the owner. The "single point of failure" problem admits a battle-tested solution; *threshold signatures*, discussed next.

Threshold Signatures. Introduced by Desmedt [18] and Desmedt and Frankel [19], a t-out-of-n threshold signature scheme is a mechanism for a group of n signatories that provides the following functionality and security guarantee: Any *quorum* of $t \leq n$ signatories may generate a valid signature σ for an arbitrary message msg, and no adversary controlling fewer than t signatories can forge a signature, i.e. it cannot produce a pair (msg', σ') such that σ' is a valid signature for msg' (provided msg' was never signed before by a quorum of t parties). In recent years, motivated by applications to cryptocurrency custody, many truly-practical protocols have been proposed for "thresholdizing" ECDSA signatures, that is, the recent proposals replace the signing algorithm with a secure *interactive* multi-party protocol involving n signatories, thus realizing the "threshold" paradigm both in functionality and security (see Section 1.1 for related work on threshold ECDSA). In fact, many companies have incorporated these protocols in their digital asset infrastructures,[1] and the total transaction volume is estimated at *trillions* of US dollars.[2]

While recent protocols effectively solve the "single-point-of-failure" problem very efficiently for ECDSA-based digital assets, the highly interactive nature of existing MPC protocols makes these solutions incompatible (or at least cumbersome to use) with so-called *cold storage*, defined next.

Cold Storage. Cold storage refers to the general principle of safeguarding the secret material underlying a digital asset (e.g. the ECDSA secret key for Bitcoin) on a platform that is disconnected from the internet, thus providing an extra layer of security against theft. For example, an ECDSA secret key written on a piece of paper constitutes a cold-storage solution (also known as a paper *wallet*). To preserve ease of use, however, most cold wallets store the secret key in a hardware device with partial computation and communication capabilities (e.g. USB stick), because, at the very least, the device is required to receive messages for signing, calculate signatures, and communicate these signatures to the world. More generally, cold-storage solutions must be as simple as possible (since physical portability may be essential and/or maintenance may be unfeasible) and they should minimize the *exposure* of the cold-storage device (here exposure is measured in connected time and total bandwidth received).

1.1 Threshold ECDSA & Cold Storage

Before we turn to our results, we discuss related work on threshold ECDSA [25], specifically as it relates to cold-storage (we refer the reader to the survey of Aumasson et al. [2] for a more thorough overview of general threshold-ECDSA protocols). As mentioned above, all protocols for threshold ECDSA require multiple rounds of communication (at least four). To make matters worse, the data sent in any given round depend on the data sent by all signatories in previous rounds. Consequently, building a wallet infrastructure that simultaneously supports threshold ECDSA (i.e. the key generation and the signing processes are distributed among many signatories) and cold storage (i.e. at least one of the signatories has limited connectivity to the internet) seems impractical given the current state of the art.

[1] We mention, e.g., Fireblocks, Unbound Security (acquired by Coinbase), Curv (acquired by Paypal) and ZenGo.

[2] Quote: "... surpassing \$2 trillion in assets transferred" retrieved from fireblocks.com (February 2023).

Pre-Signing. Beginining with Dalskov et al. [16], many recent protocols support a preprocessing mode of operation (henceforth *pre-signing*[3]) that allows the signatories to execute (parts of) the protocol before the message to be signed is known. At first, pre-signing appears as a good way to minimize exposure of a cold storage device for the following reason: pre-signing gives rise to a non-interactive signing phase which is *cheaper* than calculating a standard, non-threshold, ECDSA signature, because the message-dependant and MPC-heavy part of the signature is calculated during pre-signing. Thus, when the message for signing is presented, each signatory locally calculates their signature-share and they communicate that share to the "world"; this process is very lightweight and it does not involve any interaction.

Why the above protocols fall short on cold storage? On closer inspection, it is easy to see that pre-signing does not truly limit the exposure of the cold-storage device; it simply shifts *when* the device is exposed.[4] In other words, while pre-signing renders the signing phase non-interactive, *pre-signing itself is highly interactive*! So, the previous protocols are not particularly well suited for cold storage, because they require at least three rounds of interaction during pre-signing or signing, and each round requires heavy data processing that depends on the previous rounds. In terms of resources, per (pre-)signature, previous protocols require either *hundreds* of KB in communication complexity, e.g. for secp256k1 (the Bitcoin curve), or a large number of expensive public-key operations (10s to 100s of exponentiations in an RSA or Class Group, depending on the protocol). Concretely, for standard choice of parameters, the cold signatory consumes the following resources for calculating 10000 (pre-)signatures: depending on the choice of protocol, the "cold" signatory sends and receives *gigabytes* of data, or, it spends *minutes* to *hours* in pure computation-time, all while running an interactive (pre-)signing protocol with the online signatories.[5]

A recent proposal. The recent work of Abram et al. [1] proposes a solution to the cold-storage problem which relies on *silent preprocessing* via *pseudorandom correlation generators* (PCGs). To elaborate, similarly to OT-Extensions, PCGs allow the signatories to incur a one-time cost in order to generate practically arbitrary "ECDSA correlations". In turn, these give rise to a non-interactive signing phase where each signatory independently publishes their signature-share using their part of the ECDSA correlation. However, PCGs for ECDSA are based on fairly new cryptographic assumptions, and real-world deployment is at an early stage at the time of writing.

In this work, we propose an alternative solution based on more conventional assumptions.[6]

1.2 Our Results

We present a new threshold-ECDSA protocol Σ_{ecdsa} where one of the parties is distinguished. Furthermore, our protocol admits a *non-interactive* pre-signing phase,[7] and, for the signing phase, it suffices for the distinguished party to receive a short message from the "outside" world. We hold that Σ_{ecdsa} constitutes an attractive solution for cold storage since the non-interactivity of the pre-signing phase and the asymmetry of the signing phase inherently limit the exposure of the cold-storage device (i.e. the distinguished party).

For reducing the communication complexity of the distinguished party (i.e. the bandwidth exposure of the cold device), our protocol utilizes a new *batch-proving* technique for so-called Schnorr-style proofs, *aka* proofs arising from one-way homomorphisms. Batching is used extensively in our protocol with impressive complexity gains, especially communication-wise (c.f. Section 1.2.2). Given the numerous applications of Schnorr-style proofs,[8] we view batch-proving as a main contribution of independent interest.

[3] Pre-signing was first observed in [16] and then independently in [11, 17, 23, 24]

[4] In fact, it can be argued that pre-signing *increases* the exposure of the cold device (e.g. when pre-signatures exceed the expected future signatures).

[5] The stated values are extrapolated from the experimental results of [43] comparing the most competitive two-party protocols from [9, 12, 13, 20, 30, 31, 43] (cf. Table 2)

[6] Note that [1] makes no distinction between online and offline signatories, so it exceeds the requirements of our use case, wastefully-so, because of the computational overhead (cf. Appendix C).

[7] To handle rushing adversaries (cf. Section 2.3), the distinguished party waits to receive the outside data before it sends its own. (this does not affect the application to cold-storage since communication is inherently non-simultaneous in the real world)

[8] Schnorr-style proofs have a three-decade history in academic cryptography and there is an ongoing standardization effort [29].

1.2.1 Asymmetric Threshold ECDSA

Building on the two-party protocols of Lindell [30], MacKenzie and Reiter [32] and the n-party protocol of Canetti et al. [11], we design a new $(n+1)$-party protocol involving n online signatories $\mathcal{P}_1, \ldots \mathcal{P}_n$, dubbed the *cosignatories*, and a single offline signatory $\mathcal{P}_0$. To ease the presentation of our protocol, denoted Σ_{ecdsa}, we first describe the two-party "skeleton" (essentially the same template as [30] and [32]) between the offline party $\mathcal{P}_0$ and a single online party $\mathcal{P}_\infty$, and then we present the main ideas that give rise to our multi-party protocol.

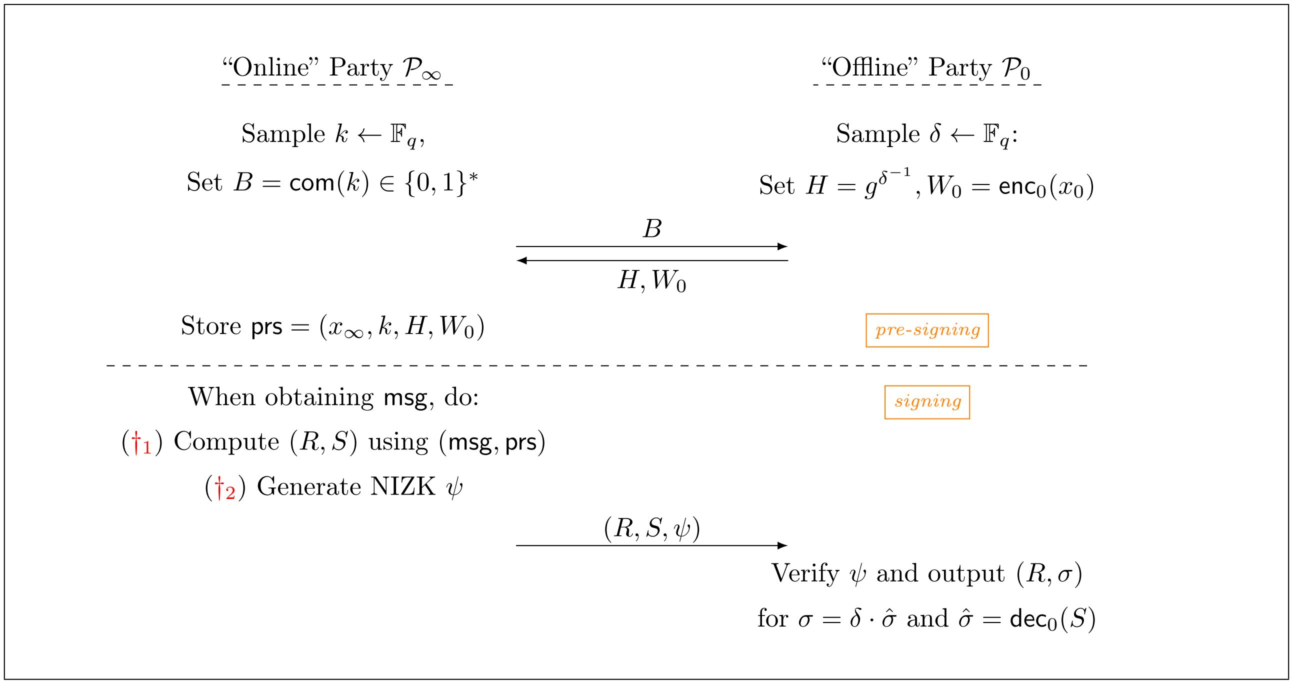

Figure 1: Threshold ECDSA, 2PC Skeleton (Σ_{ecdsa} for $n = 1$) – Write $(\mathbb{G}, q, g)$ for the group-order-generator tuple associated with ECDSA and $\mathbb{F}_q$ for the prime-order field of size q. x_0 and x_∞ denote the secret key shares of $\mathcal{P}_0$ and $\mathcal{P}_\infty$ respectively (sampled during key generation). Furthermore, enc_0 and dec_0 denote the encryption and decryption processes of an *additively-homomorphic* scheme (henceforth AHE; later we instantiate $\mathsf{enc}_0(\cdot)$ with Paillier encryption) and $\mathsf{com}(\cdot)$ denotes a commitment. The dashed line separates the pre-signing and the signing phases, and we remark that W_0 can be sent prior to pre-signing during key-generation. Notice that pre-signing is *non-interactive* since the parties' messages are independent of one another, and the signing phase is asymmetric as it boils down to $\mathcal{P}_0$ receiving a single message from $\mathcal{P}_\infty$. The main novelty of our work is showing how to "virtualize" $\mathcal{P}_\infty$ and the main challenge is realizing steps ($\dagger_1$) and ($\dagger_2$) in a distributed way. The resulting protocol inherits both non-interactivity during pre-signing and asymmetry during signing, because $\mathcal{P}_0$'s view of the protocol is essentially the same for any n.

Our MPC Protocol (Σ_{ecdsa} for $n > 2$). For the multi-party variant of our protocol, our strategy is to virtualize $\mathcal{P}_\infty$ by means of an MPC protocol. That is, we instruct the parties to *jointly* compute x_∞, k, B during key-generation and pre-signing, and (R, S, ψ) during signing. That way, the exposure of the offline party to the outside world is the same as in the two-party variant of our protocol (which we view as a tolerable baseline). The core technique for realizing the above is *party virtualization* (for $\dagger_1$) and *proof aggregation* (for $\dagger_2$). Furthermore, we present two optimization techniques with the aim of reducing the communication complexity of the protocol (thus reducing the bandwidth exposure of $\mathcal{P}_0$).

First, our protocol supports *packing*, meaning that multiple partial signatures $(\hat{\sigma}_1, \ldots, \hat{\sigma}_\lambda)$ can be loaded into a single ciphertext S (using the notation from Figure 1). Second, our protocol supports *batch-proving*, meaning that a *single* short proof ψ is used to validate the well-formedness of *many* pairs $(R_1, S_1), \ldots, (R_m, S_m)$. All of the aforementioned techniques are discussed in detail in Section 1.3.

	Communication	Computation
This Work: Pre-Signing	$3n\boldsymbol{\nu}/\lambda + 2n\boldsymbol{\gamma}$ ($n \cdot 300$B) in $2\boldsymbol{\nu}/\lambda + n\boldsymbol{\nu}/\lambda + \boldsymbol{\gamma}$ ($n \cdot 300$B) out	$\boldsymbol{N}'/\lambda + n(\boldsymbol{N}/\lambda + 3\boldsymbol{s} + \boldsymbol{s}' + 2\boldsymbol{g})$ $\leq (2n+1) \cdot \boldsymbol{N}$
This Work: Signing	$3\boldsymbol{\nu}/\lambda + 6\boldsymbol{\gamma}$ (300B) in (0B) out	$\boldsymbol{N}'/\lambda + \boldsymbol{N} + \boldsymbol{s}' + 4\boldsymbol{g}$ $\leq 3\boldsymbol{N}$
Best Computation [43]	$8\boldsymbol{\gamma}^2$ (90KB)	$0.9\boldsymbol{N} \leq 11\boldsymbol{g}$
Best Communication [43]	$16\boldsymbol{\nu} + 11\boldsymbol{\gamma}$ (4.5KB)	$14\boldsymbol{N} \leq 14\boldsymbol{N} + 11\boldsymbol{g}$
Best Comm. (Class Groups) [12]	$14\boldsymbol{\gamma}$ (500B)	$120\boldsymbol{N} \leq 4\boldsymbol{C} + 8\boldsymbol{g}$

Table 1: Costs for Offline Signatory vs Best 2PC ECDSA. We compare the offline party's costs versus the most competitive *two-party* protocols. n corresponds to the number of online signatories and the communication column displays *total* incoming (in) and outgoing (out) communication. In parentheses we report concrete estimates for secp256k1 (the Bitcoin curve) with appropriate choice of parameters. $\boldsymbol{N}$ and $\boldsymbol{N}'$ denote the unit cost of exponentiation modulo N and N^2, respectively, where N is a $\boldsymbol{\nu}$-bit RSA modulus ($\boldsymbol{s}$ and $\boldsymbol{s}'$ denote low-weight exponentiation, i.e. the bit length of the exponent is at least 10 times smaller than $\boldsymbol{\nu}$), and $\boldsymbol{C}$ denotes exponentiation in the relevant class group. When simplifying computational costs (in orange), we use the bounds $10\boldsymbol{N}' \leq \boldsymbol{C}$ (inferred from the experimental results of [43]), and the customary $\boldsymbol{N}' \leq 3\boldsymbol{N}$, $\boldsymbol{s}' \leq 3\boldsymbol{s}$, $\boldsymbol{s} \leq \boldsymbol{N}/5$. Similarly, $\boldsymbol{g}$ denotes the unit-cost of exponentiation in the ECDSA group (and $\boldsymbol{g}$ is much cheaper than either $\boldsymbol{N}$ and $\boldsymbol{C}$) and $\boldsymbol{\gamma}$ is the corresponding bit-length. Parameter $\lambda \in \mathbb{N}$ represents the *packing number* (see Section 1.3) and it is assumed $\lambda \geq 3$ in the concrete estimations. All costs are amortized over the number of signatures (or pre-signatures). For [12, 43], we report only one of either signing or pre-signing (the most expensive). For the round complexity, we note that all protocols consist of three uni-directional rounds (this is not reported in the table).

Comparison. For the two-party variant of our protocol (when $n = 1$, the costs for the solitary online party are essentially identical to those of $\mathcal{P}_0$), our protocol is by far the most communication efficient (the only protocol that achieves somewhat comparable efficiency is the Class group protocol of [12]), and, in computation, our protocol is x5 more efficient compared to the most communication-efficient protocols.

For the case $n > 1$, the complexity of the online parties is dominated by the *joint* computation of the short message that $\mathcal{P}_0$ receives. However, as we shall see in Section 1.3, there is a lot of flexibility in the choice of MPC protocol for performing this computation, because the desired functionality is almost identical to the ECDSA functionality. Therefore, the complexity costs for the cosignatories are very close to those of any state-of-the-art threshold-ECDSA protocol (the one chosen to instantiate the so-called virtual party). In our experiments (Appendix C),[9] we opted for the protocol of Canetti et al. [11] (CMP) which is somewhat expensive computation-wise.

Security & Composability. Most protocols in the literature show security against so-called *static* adversaries, where the corruption pattern of the attacker is fixed at the beginning of the execution. In contrast, in this work, we show that our protocol achieves security against adaptive adversaries, i.e. the adversary may choose which parties to corrupt as the protocol evolves and the identity of the corrupted parties may be chosen adaptively as a function of the adversary's view. Furthermore, we show that our protocol is composable in the UC framework and it realizes the ideal threshold-signatures functionality $\mathcal{F}_{\mathsf{tsig}}$ from [11], i.e. even when it is composed arbitrarily with other components of some larger system, the protocol emulates the ideal functionality.

Theorem (Informal)**.** *Under suitable cryptographic assumptions, protocol* Σ_{ecdsa} *UC-realizes* $\mathcal{F}_{\mathsf{tsig}}$ *against adaptive adversaries.*

[9] As a sample from our experimental findings, we mention the following. For calculating 10000 pre-signatures and $n = 2$ (i.e. two online parties),[10] our protocol takes less than a minute of CPU time and roughly 3 megabytes of data are exchanged between each online party and $\mathcal{P}_0$. For the signing phase, the online parties spend less than two minutes of CPU time to aggregate the approximately 5MB "payload" which they send to $\mathcal{P}_0$, and $\mathcal{P}_0$ spends less than a minute of CPU time to process the data and output the signatures. (In our experiments, the online parties run the CMP protocol among themselves prior to the aggregation phase which slows downs the signature-generation process by approximately 300ms per signature; this choice is arbitrary and CMP may be substituted for a more computation-friendly protocol, e.g. DKLs19 [20], if so desired.)

To show the above, we simplify and generalize the proof technique of [11] where the indistinguishability of the UC simulation is reduced to the unforgeability of the underlying non-threshold signature scheme. As a corollary of independent interest, we find that all *non-pathological* threshold-signatures protocols effectively UC-realize the $\mathcal{F}_{\mathsf{tsig}}$ functionality (a protocol is non-pathological if forging signatures is unfeasible for all adversaries corrupting fewer parties than the minimum signing threshold). We refer the reader to Section 1.3.4 for further details.

1.2.2 New Batch-Proving Technique

We present a new batching technique for proving in zero-knowledge (ZK) that many *instances* $X_1, \ldots, X_m \in \{0,1\}^*$ belong to a certain language. Specifically, we are interested in languages that arise from one-way homomorphisms (discussed next) and our batching technique yields proofs with *zero* communication overhead, i.e the proof size is independent of m. (The computation gains are also noteworthy, but more modest, cf. Table 2.)

Schnorr-Style Protocols. Almost all ZK proofs herein can be cast as Fiat-Shamir transforms of ZK protocols for group homomorphism (referred to as *Schnorr Protocols* in this document). In line with Bangerter [3] and Maurer [35], we define Schnorr protocols abstractly (c.f. Section 2.5) as protocols for proving that the pre-image $w \in \mathbb{H}$ of a group element $X \in \mathbb{G}$ by some homomorphism $\phi : \mathbb{H} \to \mathbb{G}$ is "close" to a subset $\boldsymbol{R} \subseteq \mathbb{H}$. Throughout the paper, we will be using additive notation for $\mathbb{H}$ and multiplicative notation for $\mathbb{G}$. So, assuming that $\mathbb{H}$ is endowed with a suitable distance function, write $\boldsymbol{R}' = \{\alpha \in \mathbb{H} \text{ s.t. } \alpha \text{ is "close" to } \boldsymbol{R}\}$, and observe that every ϕ and $\boldsymbol{R}$ gives rise to the following three-round Prover-Verifier protocol parametrized by $\boldsymbol{E} \subseteq \mathbb{Z}$ (or a *non-interactive zero-knowledge proof*, aka NIZK, via the Fiat-Shamir transform). Letting $(X; w)$ denote the instance-witness pair, i.e. $X = \phi(w)$:

1. The Prover sends $A = \phi(\alpha)$ to the Verifier for random $\alpha \leftarrow \boldsymbol{R}'$.
2. The Verifier replies with "challenge" $e \leftarrow \boldsymbol{E}$.

   For the NIZK, the Prover applies the Fiat-Shamir transform to locally calculate e.
3. The Prover answers the challenge by sending $z = \alpha + ew \in \mathbb{H}$.

   The Verifier accepts if $g^z = A \cdot X^e$ and $z \in \boldsymbol{R}'$.

Generic Batch-Proving. With the aim of batching many instances $X_1, \ldots, X_m$ into a single protocol/proof, we identify the following generic transformation (the related works of Gennaro et al. [26] and Thyagarajan et al. [41] can be cast as special cases of the generic template below). We dub the resulting protocol an *m-batch* Schnorr protocol.

1. The Prover sends $A = \phi(\alpha)$ to the Verifier for random $\alpha \leftarrow \boldsymbol{R}'$.
2. The Verifier replies with challenges $(e_1, \ldots, e_m) \leftarrow \boldsymbol{E}^m$.
   (a) In [26], $(e_1, \ldots, e_m) = (e^1, e^2, \ldots, e^m)$ for a random $e \leftarrow \boldsymbol{E}$
   (b) In [41], $(e_1, \ldots, e_m) \leftarrow \{0,1\}^m$, i.e. the e_i's are uniform independent bits.
   (c) **In this work**, we sample $(e_1, \ldots, e_m)$ such that each $e_i \leftarrow \boldsymbol{E}$ is independently drawn.
3. The Prover answers the challenge by sending $z = \alpha + \sum_{i=1}^m e_i w_i \in \mathbb{H}$.

   The Verifier accepts if $g^z = A \cdot \prod_{i=1}^m X_i^{e_i}$ and $z \in \boldsymbol{R}'$.

Theorem (Informal)**.** *If $w_i \notin \boldsymbol{R}'$ for some $i \in [m]$, then the verification process of the associated m-batch Schnorr protocol fails with overwhelming probability, assuming the tuple $(\phi, \boldsymbol{E}, \boldsymbol{R})$ is "well-behaved".*

In Section 1.3.2, we define "well-behavedness" and we give a proof-sketch of the above.

Remark 1.1. The main novelty of the above is twofold. On one hand, we identify a generic template for batching many Schnorr protocols/proofs that captures the previous techniques. On the other hand, we propose a concrete instantiation (where the e's are sampled independently at random) and we prove that it yields good protocols/proofs.

	Comm.	Prover Comp.	Verifier Comp.
MacKenzie and Reiter [32]	$m \cdot 5\boldsymbol{\nu}$	$m \cdot (\boldsymbol{N}' + \boldsymbol{N} + 2\boldsymbol{s})$ $\approx m \cdot (4\boldsymbol{N} + 2\boldsymbol{s})$	$m \cdot (\boldsymbol{N}' + \boldsymbol{N} + \boldsymbol{s}' + 2\boldsymbol{s})$ $\approx m \cdot (4\boldsymbol{N} + 5\boldsymbol{s})$
Thyagarajan et al. [41]	$\kappa \cdot 3\boldsymbol{\nu}$	$\kappa \cdot \boldsymbol{N}'$ $\approx \kappa \cdot 3\boldsymbol{N}$	$\kappa \cdot \boldsymbol{N}'$ $\approx \kappa \cdot 3\boldsymbol{N}$
Batch-Proving (**Our Work**)	$5\boldsymbol{\nu}$	$\boldsymbol{N}' + \boldsymbol{N} + (m+1) \cdot \boldsymbol{s}$ $\leq (0.2m + 4) \cdot \boldsymbol{N}$	$\boldsymbol{N}' + \boldsymbol{N} + \boldsymbol{s} + m \cdot (\boldsymbol{s}' + \boldsymbol{s})$ $\leq (0.8m + 3) \cdot \boldsymbol{N}$

Table 2: Batch-Proving Comparison. In the ROM, for security parameter κ, comparison of [32, 41] with our work for proving that the plaintext values of m Paillier ciphertext lie in a given range. The values $\boldsymbol{\nu}$, $\boldsymbol{s}, \boldsymbol{s}', \boldsymbol{N}, \boldsymbol{N}'$ are as in Table 1, and the target range is $[-2^{\nu/10}, 2^{\nu/10}]$. The protocol of Thyagarajan et al. [41] achieves constant soundness and requires amplification via parallel repetition; hence the dependency on κ.

Comparison. In Table 2, we provide a comparison of our flavor of batch-proving vs the baseline protocol of MacKenzie and Reiter [32], and the batch-proving protocol of Thyagarajan et al. [41]; the communication complexity of our scheme compares favorably by orders of magnitude for any batch size (we note that [41] overtakes our protocol in computation for large batches). Finally, our table does not include the batch-protocol of Gennaro et al. [26] because the successive powers $e, e^2, e^3, \ldots, e^m$ are not well-suited for proving range (there is a 2^m-blowup in the target range).

1.3 Technical Overview

In this section, we give thorough overview of our techniques, including informal theorem-statements and proof-sketches. We begin by presenting our threshold-ECDSA protocol Σ_{ecdsa} in more detail (omitting non-essential details). Then, we discuss batch-proving and packing. Finally, we conclude with the security analysis and we state our main security claim for Σ_{ecdsa}.

1.3.1 Party Virtualization & Proof Aggregation

Let $(\mathbb{G}, g, q)$ denote the group-generator-order tuple for ECDSA and write $\mathbb{F}_q$ for the finite field of prime order q. For this high-level overview, we will abuse notation and write $R \cdot x \in \mathbb{F}_q$ to denote the field element obtained by projecting[11] $R \in \mathbb{G}$ in $\mathbb{F}_q$ and multiplying by $x \in \mathbb{F}_q$. For secret key $x \in \mathbb{F}_q$, letting $m \in \mathbb{F}_q$ denote the hash of a desired message, we recall that ECDSA signatures have the form (R, σ) for $\sigma = k(m + R \cdot x) \in \mathbb{F}_q$ and $R = g^{k^{-1}} \in \mathbb{G}$. Finally, we define the $\mathtt{tecdsa}_{(\cdot)}$ functionality s.t.

$$\mathtt{tecdsa}_g : (x_1, \ldots, x_n) \mapsto (R, k_i, \chi_i)_{i \in [n]}$$

where $R = g^{(\sum_i k_i)^{-1}}$ and $(\sum_i k_i)(\sum_i x_i) = \sum_i \chi_i$, and $\{x_i\}_i$ are additive shares of the secret key. Using the outputs of $\mathtt{tecdsa}_g$, observe that the following values can be summed up to obtain an ECDSA signature: $\{\sigma_i\}_{i \in [n]}$ for $\sigma_i = k_i m + R \cdot \chi_i$.

Fact. *All multi-party ECDSA protocols from Section 1.1 compute the* $\mathtt{tecdsa}_{(\cdot)}$ *functionality .*

Our Protocol (Σ_{ecdsa}). The following key observation summarises the signing process of our protocol. By calling $\mathtt{tecdsa}_{(\cdot)}$ on point H provided by $\mathcal{P}_0$ (instead of g), the online parties can calculate an encryption $S = \mathsf{enc}_0(\hat{\sigma})$ of a "partial" signature $\hat{\sigma}$ such that the pair $(R, \sigma) = (H^{k^{-1}}, \delta \cdot \hat{\sigma})$ conforms to the signature format of ECDSA, where R is common output of $\mathtt{tecdsa}_H$ and S is jointly calculated using the homomorphic properties of $\mathsf{enc}_0(\cdot)$. Thus, by obtaining (R, S), $\mathcal{P}_0$ can finalize $\hat{\sigma}$ into the "full" signature σ (cf. Figure 2).

[11] We recall that (almost) all group elements $R \in \mathbb{G}$ may be viewed as pairs of field elements (a, b) with $a, b \in [0, q-1]$.

FIGURE 2 (Simplified Variant of $\Sigma_{\mathtt{ecdsa}}$ w/o Proofs.)

KeyGen & PreSign. $\mathcal{P}_0$ sends $W_0 = \mathsf{enc}_0(x_0)$ and $H = g^{\delta^{-1}} \in \mathbb{G}$ to $\mathcal{P}_1, \ldots, \mathcal{P}_n$.

The above happens during key generation and pre-signing respectively.

Sign. When obtaining msg $\in \{0,1\}^*$ for signing, set $m = \mathrm{HASH}(\mathrm{msg})$ and do:

1. $\mathcal{P}_1, \ldots, \mathcal{P}_n$ run a protocol for computing $\mathtt{tecdsa}_H(x_1, \ldots, x_n)$. They obtain $(R, k_i, \chi_i)_{i \in [n]}$.
2. Each signatory $\mathcal{P}_i$ homomorphically evaluates

$$S_i = \mathsf{enc}_0(k_i m + R \cdot \chi_i) \oplus (R \cdot k_i \odot W_0) = \mathsf{enc}_0(k_i m + R \cdot (\chi_i + k_i x_0))$$

and they send (R, S) to $\mathcal{P}_0$, where S is the aggregate ciphertext $S = \oplus_{i=1}^n S_i$ to $\mathcal{P}_0$.

3. $\mathcal{P}_0$ calculates $\hat{\sigma} = \mathsf{dec}_0(S)$ and outputs the ECDSA signature (R, σ) for $\sigma = \delta \cdot \hat{\sigma}$.

Figure 2: Simplified Variant of $\Sigma_{\mathtt{ecdsa}}$ w/o Proofs. We recall that $\mathcal{P}_0$ denotes the offline signatory and $\mathcal{P}_1, \ldots, \mathcal{P}_n$ denote the cosignatories and $\mathsf{enc}_0(\cdot)$ denotes $\mathcal{P}_0$'s AHE. Note that $\oplus$ and $\odot$ denote the homomorphic operations of addition and multiplication by scalar, respectively. We assume that $x_0, x_1, \ldots, x_n$ are sampled during the key-generation phase and the ECDSA public key is set to $\mathsf{pk} = g^{\sum_{i=0}^n x_i}$ (At this stage, we will ignore $\mathsf{com}(k)$ from Figure 1 for this informal presentation). Observe that Item 2 in Figure 2 is equivalent to $\mathcal{P}_0$ receiving a message from a single *virtual* party and this message is independent of the number of parties. For $n = 1$, the above uses the same template as the two-party protocols of MacKenzie and Reiter [32] or Lindell [30].

Malicious Security & Proof Aggregation. For malicious security, the pair (R, S) is accompanied by a short ZK proof ψ validating that S is well-formed and it contains the right $\hat{\sigma}$. Notice that each S_i may be viewed as the image of the following homomorphism, letting $(\beta_i, \gamma_i) = (k_i m + R \cdot \chi_i, R \cdot k_i)$,

$$\phi : (\beta_i, \gamma_i) \mapsto (\beta_i \odot \mathsf{enc}_0(1)) \oplus (\gamma_i \odot W_0). \tag{1}$$

So, since ϕ is a homomorphism, it follows that each $\mathcal{P}_i$ can prove well-formedness of S_i by generating a Schnorr proof, as discussed in Section 1.2.2; this approach incurs $O(n)$ communication overhead because each $\mathcal{P}_i$ provides its own proof. To avoid the communication penalty, we propose the generic aggregation process described below. This way, $\mathcal{P}_1, \ldots, \mathcal{P}_n$ calculate a single short proof ψ validating that the aggregated ciphertext S is well-formed, and the size of ψ is independent of n. So, using the notation from Section 1.2.2, we define:

1. The Provers $\mathcal{P}_1, \ldots, \mathcal{P}_n$ *jointly* calculate $A = \phi(\alpha) = \prod_{i=1}^n \phi_i(\alpha_i)$ where each $\mathcal{P}_i$ chooses α_i.
2. The Verifier replies with $e \leftarrow \boldsymbol{E}$.

   For the NIZK, the Provers apply the Fiat-Shamir transform to locally calculate e.
3. The Provers *jointly* calculate and send $z = \sum_{i=1}^n z_i$ where each $\mathcal{P}_i$ calculates $z_i = \alpha_i + e w_i$.

   Output. $\psi = (A, e, z)$

It is easy to see that the Provers cannot cheat in the above process, even if all of them collude, because, from the Verifier's point of view, the aggregated proof ψ inherits all the security properties of the non-aggregated proof. Furthermore, the aggregation process does not interfere with batch-proving and thus aggregate proofs are "batchable" when many instances are presented (it is easy to see this fact). In summary, letting m denote the number of instances for batching, our approach yields a multiplicative $(n \cdot m)$-improvement in proof size compared to the non-batch, non-aggregate, baseline. (Looking ahead, we will further improve the communication complexity using the *Packing* optimization described in Section 1.3.3.)

Next, we give a high-level overview of the security analysis for batch-proving.

1.3.2 Batch-Proving

Hereafter, $\phi : \mathbb{H} \to \mathbb{G}$ denotes an arbitrary group homomorphism and let $\boldsymbol{R} \subseteq \mathbb{H}$ and $\boldsymbol{E} \subseteq \mathbb{Z}$. Further, assuming $\mathbb{H}$ is endowed with a suitable distance function, let $\boldsymbol{R}'$ denote the subset of $\mathbb{H}$ of values which are close to $\boldsymbol{R}$, i.e. $\boldsymbol{R}' = \{h \in \mathbb{H} \text{ s.t. } h \text{ is close to } \boldsymbol{R}\}$. In this section, we define sufficient conditions for the security of batch-proving, i.e. we elaborate on the "well-behavedness" of the tuple $(\phi, \boldsymbol{E}, \boldsymbol{R})$ and we give a proof sketch that the associated batch protocol is secure.

Definition (Informal)**.** Let Π denote the m-batch protocol for $(\phi, \boldsymbol{E}, \boldsymbol{R})$, and define the following conditions.

1. $\boldsymbol{V}$*-Extractability.* $\forall$ PPTM $\mathcal{A}$, If:
   (a) $(\tau_1, \ldots, \tau_{m+1}) \leftarrow \mathcal{A}$ such that $\tau_j = (A, \vec{e}_j, z_j)$ and all τ_j's begin with $A \in \mathbb{G}$,
   (b) τ_j is a valid transcript for the associated Schnorr protocol, for all $j \in [m+1]$,
   (c) $(\vec{e}_1, \ldots, \vec{e}_{m+1}) \in \boldsymbol{V}$,

   Then, $\exists$ PPTM $\mathcal{E}$ s.t. $(w_1, \ldots, w_m) \leftarrow \mathcal{E}(\tau_1, \ldots, \tau_{m+1})$ and $\forall j$, $\phi(w_j) = X_j$.
2. *Robustness.* For all $j \in [m]$, it holds that $\Pr_{\vec{e}_{j+1} \leftarrow \boldsymbol{E}^m}[(\vec{e}_1, \ldots, \vec{e}_j, \vec{e}_{j+1}) \in \boldsymbol{V} \mid (\vec{e}_1, \ldots, \vec{e}_j) \in \boldsymbol{V}] \approx 1$.
3. *Unpredictability.* For all $\alpha \in \mathbb{H}$, if $w_j \notin \boldsymbol{R}'$ for some $j \in [m]$, then $\Pr_{\vec{e} \leftarrow \boldsymbol{E}^m}[\alpha + \sum_{k=1}^m e_k w_k \in \boldsymbol{R}'] \approx 0$.
4. *Collision Resistance.* $\forall$ PPTM $\mathcal{A}$, it holds that $\Pr_{w \neq w' \leftarrow \mathcal{A}}[\phi(w) = \phi(w')] \approx 0$.

Theorem (Batch Security – Informal)**.** *Assume there exists* $\boldsymbol{V}$ *such that the tuple* $(\phi, \boldsymbol{E}, \boldsymbol{R})$ *satisfies Extractability, Robustness, Unpredictability and Collision Resistance. If* $w_i \notin \boldsymbol{R}'$ *for some* $i \in [m]$*, then the verification process of the associated* m*-batch Schnorr protocol fails with overwhelming probability.*

Proof Sketch. Let $\mathcal{A}$ denote an adversary passing the verification for a faulty $w_i \notin \boldsymbol{R}'$. We define the following experiment between $\mathcal{A}$ and a simulator $\mathcal{S}$ where $\mathcal{S}$ plays the role of the verifier. After $\mathcal{A}$ hands out its first message $A \in \mathbb{G}$, the simulator forks the protocol into $r \in \mathbb{N}$ parallel sessions and in the k-th session $\mathcal{S}$ returns an independently sampled Verifier-challenge $\vec{e}_k \leftarrow \boldsymbol{E}^m$ to $\mathcal{A}$. The experiment concludes when $\mathcal{A}$ hands over the third-round message z_k for every session ($\mathcal{A}$ may return $\perp$ if it decides to abort). The theorem follows from the following observations. (i) By concentration inequality, for carefully chosen $r \in \mathbb{N}$, at least $m+2$ of the transcripts are valid; wlog assume that the first $m+2$ transcripts are valid. (ii) By $\boldsymbol{V}$-Extractability and Robustness, we can extract all the pre-images $\alpha, \vec{w} = (w_1, \ldots, w_m)$ using the first $m+1$ transcripts. (iii) By Unpredictability, for all $k \geq m+2$, it holds that $\hat{z}_k = \alpha + \vec{e}_k \cdot \vec{w} \notin \boldsymbol{R}'$ (recall that at least one of the w's is out of range). (iv) Since $\tau_{m+2} = (A, \vec{e}_{m+2}, z_{m+2})$ is a valid transcript, it holds that $z_{m+2} \in \boldsymbol{R}'$. (v) Since ϕ is a homomorphism, it holds that $\phi(z_{m+2}) = \phi(\hat{z}_{m+2})$.

The last item yields a contradiction with Collision Resistance. □

Or threshold ECDSA protocol makes extensive use of batch-proofs resulting from the Pedersen homomorphism (which gives rise to so-called Pedersen commitments, aka Fujisaki-Okamoto commitments [22]). As we shall see next, showing that Pedersen commitments satisfy the above conditions is non-trivial, and we view this contribution as an additional technical novelty of our work (for the informal technical overview, we focus on extractability).

Definition (Pedersen Commitments – Informal)**.** Let $\mathbb{P}$ be group and let $t \in \mathbb{P}$ and $s \in \langle t \rangle$ denote random elements in $\mathbb{P}$ and $\langle t \rangle$ (the group generated by t) respectively. Let $\phi : \mathbb{Z} \times \mathbb{Z} \to \mathbb{P}$ denote the homomorphism $\phi(w, \rho) = t^\rho s^w$. We say that C is a *Pedersen commitment* of $w \in \mathbb{Z}$, if $C = \phi(w, \rho)$, for some $\rho \in \mathbb{Z}$.

For arbitrary $\boldsymbol{E}, \boldsymbol{R}$, let Π denote the m-batch protocol for tuple $(\phi, \boldsymbol{E}, \boldsymbol{R})$. Define $\boldsymbol{V} \subseteq 2^{\boldsymbol{E}}$ such that $(\vec{e}_1, \ldots, \vec{e}_i) \in \boldsymbol{V}$ if there exists $\vec{e}_{i+1}, \ldots, \vec{e}_{m+1} \in \boldsymbol{E}^m$ such that $\det(E) \neq 0$ and $\det(E)$ is coprime with $|\mathbb{P}|$, where:

$$E = \begin{pmatrix} 1 & \vec{e}_1 \\ \vdots & \vdots \\ 1 & \vec{e}_{m+1} \end{pmatrix} = \begin{pmatrix} 1 & e_{1,1} & \ldots & e_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & e_{m+1,1} & \ldots & e_{m+1,m} \end{pmatrix}.$$

Theorem (Batch Pedersen PoK - Informal)**.** *If the strong-RSA assumption holds in* $\mathbb{P}$ *then* Π *is* $\boldsymbol{V}$*-extractable.*

Proof Sketch. Recall the strong-RSA assumption: $\Pr_{t \leftarrow \mathbb{P}}[(c,d) \leftarrow \mathcal{A}(\mathbb{P},t) \text{ s.t. } c^d = t \wedge d \notin \{-1,1\}] \approx 0$, for every PPTM $\mathcal{A}$. We describe reduction from Extractability to Strong RSA. So, we assume that Extractability does not hold and we will find a non-trivial root for RSA challenge $t \leftarrow \mathbb{P}$. First, the reduction chooses $s = t^\lambda$ where $\lambda \approx |\mathbb{P}|^2$ (the size of λ is crucial for the reduction). Next, letting $C_1, \ldots, C_m \in \mathbb{P}$ denote the m Pedersen instances to be batched, $\mathcal{A}$ generates $m+1$ valid transcripts $\tau_1 = (A, \vec{e}_1, (y_1, z_1)), \ldots, \tau_{m+1} = (A, \vec{e}_{m+1}, (y_{m+1}, z_{m+1}))$ for $m+1$ where $s^{z_i} t^{y_i} = A \cdot \prod_j C^{e_{i,j}}$ for every i. Further, assume that $(\vec{e}_1, \ldots, \vec{e}_{m+1}) \in \boldsymbol{V}$ and write F for the inverse of E over the *rational* numbers $\mathbb{Q}$ (the reduction cannot invert in $\mathbb{Z}_{|\mathbb{P}|}$ because it does not know the order of the group – strong RSA is easy otherwise). Notice that if $F \cdot \vec{y} \in \mathbb{Z}^{m+1}$ and $F \cdot \vec{z} \in \mathbb{Z}^{m+1}$, then $(w_i, \rho_i) = (F_i \cdot \vec{z}, F_i \cdot \vec{y}) \in \mathbb{Z}^2$ is a valid decommitment for C_i where F_i denotes the i-th row of F, so exctractability is not violated in this case. Else, write $F = \hat{E}/\det(E)$ for some integer matrix $\hat{E}$ (standard linear-algebra fact) and observe that $t^{\alpha\lambda+\beta} = C_i^{\det(E)}$ for $\alpha = \hat{E}_i \cdot \vec{z}$ and $\beta = \hat{E}_i \cdot \vec{z}$, letting $\hat{E}_i$ denote the i-th row of $\hat{E}$. To conclude, we will produce a suitably-chosen pair (R,d) such that $R^d = t$ which breaks strong RSA. First, note that $\det(E)$ does not divide $\gamma = \alpha\lambda + \beta$ because $\lambda \approx |\mathbb{P}|^2$ is information-theoretically hidden and λ is implicitely reduced modulo the order of the group. Consequently, letting $b = \gcd(\det(E), \gamma)$, define (R,d) such that $d = \det(E)/b$ and $R = t^u C_i^v$ where (u,v) are the Bézout coefficients[12] of $\det(E)$ and γ, and conclude that $d \notin \{-1,1\}$ and

$$t = t^{\frac{u\det(E)+v\gamma}{b}} = t^{ud}(t^{\gamma/b})^v = t^{ud} \cdot (C_i^d)^v = (t^u C_i^v)^d.$$

□

Remark 1.2. In subsequent sections, we instantiate Pedersen commitments in RSA groups.

1.3.3 Packing

In general, *packing* refers to the folklore principle of storing many (small) values in a single (large) unit of space, e.g. to encrypt $w_1, \ldots, w_\lambda \in \{0,1\}^*$ and save space, it is preferable to store a single ciphertext $C = \mathsf{enc}(\sum_{j=1}^m 2^{\tau(j-1)} w_j)$ for suitable $\tau \in \mathbb{N}$, rather than λ distinct ciphertext for each w (assuming the bit-length of the plaintext-space is larger than $\sum_j |w_j|$). We show that our protocol enjoys a performance improvement thanks to packing, and total communication is reduced by a multiplicative factor λ (e.g. $\lambda \geq 3$ for secp256k1 for standard choice of parameters).

However, packing introduces a number of design and security challenges, discussed next.

Naive Pedersen Packing. Unfortunately, Pedersen commitments do not support the natural approach for packing. Recall the Pedersen homomorphism $\phi(w,\rho) = t^\rho s^w$ and consider the *naive* Pedersen packing associated with the homomorphism $\phi'(\vec{w},\rho) = \phi(\sum_{j=1}^\lambda 2^{\tau(j-1)} w_j, \rho)$. Observe that $\phi'((0,1,0,\ldots),\rho) = \phi'((2^\tau,0,0,\ldots),\rho)$, i.e. ϕ_0 does not satisfy collision resistance, and thus the Schnorr protocol associated with ϕ' is insecure.[13] To overcome this issue, we introduce Pedersen multi-commitments.

Definition (Multi-Pedersen – Informal)**.** Let $\mathbb{P}$ be group and let $t \in \mathbb{P}$ and $s_1, \ldots, s_\lambda \in \langle t \rangle$ denote random elements in $\mathbb{P}$ and $\langle t \rangle$ respectively. Let $\phi_0 : \mathbb{Z}^\lambda \times \mathbb{Z} \to \mathbb{P}$ denote the homomorphism $\phi_0(\vec{w},\rho) = t^\rho \prod_{j=1}^\lambda s^{w_j}$. We say that C is a *Pedersen multi-commitment* of $\vec{w} \in \mathbb{Z}^\lambda$, if $C = \phi(\vec{w},\rho)$, for some $\rho \in \mathbb{Z}$.

We conclude this section by showing that ϕ_0 is collision resistant. The proof is a straightforward generalization of the so-called binding property for standard Pedersen commitments.

Theorem (Multi-Ped. Collision – Informal)**.** *If the order assumption holds in $\mathbb{P}$, then ϕ_0 is collision-resistant.*

Proof Sketch. Recall the order assumption: $\Pr_{t \leftarrow \mathbb{P}}[d \leftarrow \mathcal{A}(\mathbb{P},t) \text{ s.t. } t^d = 1 \wedge d \notin \{0\}] \approx 0$, for every PPTM $\mathcal{A}$ (note that strong RSA implies the order assumption [4]). Similarly to the proof of extractability, we define a reduction from collision resistance to the order assumption. For all $j \in [\lambda]$, it is assumed that $s_j = t^{\lambda_j}$ for random λ_j and the λ's are handed to the reduction. Let $(\vec{w},\rho,\vec{w}',\rho') \leftarrow \mathcal{A}$ such that $\phi_0(\vec{w},\rho) = \phi_0(\vec{w}',\rho')$. Deduce that $t^{\vec{\lambda}\cdot(\vec{w}-\vec{w}')+(\rho-\rho')} = 1$ and thus $d = \vec{\lambda} \cdot (\vec{w} - \vec{w}') + (\rho - \rho')$ breaks the order assumption. To see why, notice that $d \neq 0$, with high probability, since the λ's are random. □

[12]For $x, y \in \mathbb{N}$, the Bézout coefficients $u, v \in \mathbb{Z}$ satisfy $ux + yv = \gcd(x,y)$.

[13]In fact, naive packing may lead to key leakage in the ECDSA protocol, regardless of batching. We refer to [34, 42] for bit-probing attacks on Paillier-based threshold ECDSA.

1.3.4 Security Analysis

We conclude the high-level technical overview with the security analysis of our threshold ECDSA protocol. Our key technique (from [11]) is to show that our protocol UC-realizes $\mathcal{F}_{\text{tsig}}$ by way of reduction to the assumed unforgeability of the underlying non-threshold scheme. So, let SGN denote an arbitrary signature scheme (not-necessarily-ECDSA) and recall that SGN is a threetuple of algorithm for (i) generating public/private key-pairs, (ii) signing messages, and (iii) verifying signatures. We begin by defining signing oracles and unforgeability.

Definition (Unforgeability – Informal)**.** let $\mathcal{G}$ denote a signature oracle for SGN, i.e. $\mathcal{G}$ is a PPTM such that:

1. Upon activation, $\mathcal{G}$ samples a secret/public key-pair $(\mathsf{sk}, \mathsf{pk})$ as prescribed by SGN and returns pk.
2. When queried on a message msg $\in \{0, 1\}$, $\mathcal{G}$ returns a signature σ (NB. σ is a function of msg and sk).[14]

We say that SGN is $\mathcal{G}$-unforgeable if the following holds for every adversary $\mathcal{A}$ interacting with $\mathcal{G}$. For any message $m \in \{0, 1\}^*$, if m was never queried to $\mathcal{G}$ by $\mathcal{A}$, then the probability that $\mathcal{A}$ outputs a valid signature for m is negligible. Intuitively, SGN is $\mathcal{G}$-unforgeable if $\mathcal{G}$ is not useful for forging signatures.

MPC & Adversarial Model. Write Σ for an n-party protocol computing SGN, i.e. Σ is an interactive protocol between parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ such that Σ emulates the key-generation, signature, and verification algorithms of SGN. As usual, there is an adversary $\mathcal{A}$ corrupting a subset of parties, and the corrupted parties are utterly controlled by $\mathcal{A}$, e.g. they can send any message of $\mathcal{A}$'s choosing. In this work, we assume that $\mathcal{A}$ corrupts parties adaptively, i.e. $\mathcal{A}$ decides which parties to corrupt dynamically as the protocol evolves, and $\mathcal{A}$ may even decide to *decorrupt* certain parties. Security of Σ is defined in the UC-framework, discussed next.

Universal Composability (UC). Consider the following two experiments in the presence of the so-called *environment* $\mathcal{Z}$; the purpose of $\mathcal{Z}$ is to distinguish between the two experiments. (**Real**) The first experiment corresponds to the actual, i.e. real, execution of Σ where the honest parties interact with the adversary. (**Ideal**) The second experiment corresponds to the following idealized process: All parties except $\mathcal{Z}$ interact by means of an *incorruptible* trusted party $\mathcal{F}$, dubbed the ideal functionality, according to the specifications of $\mathcal{F}$.

In both experiments, $\mathcal{Z}$ has full control over the inputs of the $\mathcal{P}_i$'s, i.e. $\mathcal{Z}$ chooses the messages for signing, and $\mathcal{Z}$ can see the outputs of the $\mathcal{P}_i$'s, i.e. the resulting signatures. Furthermore, we assume that $\mathcal{Z}$ has direct communication with the adversary, and we write $\mathcal{S}$ for the ideal adversary to avoid confusion. Turning to the security definition, intuitively, Σ is secure if $\mathcal{Z}$ cannot tell the difference between real and ideal experiments, i.e. $\mathcal{Z}$'s output is almost identically distributed in both experiments. Slightly more formally, letting $\text{EXEC}^{\mathsf{real}}_{\mathcal{Z},\Sigma}$ and $\text{EXEC}^{\mathsf{ideal}}_{\mathcal{Z},\mathcal{F}}$ denote the output of $\mathcal{Z}$ in the real and ideal experiments respectively, we say that Σ UC-realizes $\mathcal{F}$ if:

$$\forall A \exists S \, \forall \mathcal{Z} \text{ it holds that } \text{EXEC}^{\mathsf{real}}_{\mathcal{Z},\Sigma} \equiv \text{EXEC}^{\mathsf{ideal}}_{\mathcal{Z},\mathcal{F}}. \tag{2}$$

Ideal functionality. When choosing which functionality to realize, the most "natural" ideal functionality, $\mathcal{F}_{\mathsf{SGN}}$, simply runs the code of SGN. In other words, $\mathcal{F}_{\mathsf{SGN}}$ samples sk, pk as prescribed by SGN and interacts with the parties as a signing oracle, similar to $\mathcal{G}$ above. However, as it is argued in [11, 33], this natural approach has certain disadvantages. For one, it rules out many protocols that are "good enough", i.e. maybe Σ only outputs biased signatures (e.g. the first bit of σ is zero), and this bias is inconsequential to the security of the protocol. In this case, Σ does *not* realize $\mathcal{F}_{\mathsf{SGN}}$ even though Σ is a "good" protocol. Second, UC-realizing $\mathcal{F}_{\mathsf{SGN}}$ may require instantiating the protocol with UC-secure NIZKs and these incur substantial overhead, even in the random oracle model, because, e.g., the Fiat-Shamir transform gives way to the Fischlin transform [21] which is more expensive in both computation and communication. Finally, adaptive security (one of the desiderata for our protocol) is notoriously hard and prohibitively expensive when realizing $\mathcal{F}_{\mathsf{SGN}}$, and it generally requires sophisticated cryptographic tools like non-committing encryption.

In this work, to circumvent the above limitations, we opt for the ideal threshold-signatures functionality $\mathcal{F}_{\mathsf{tsig}}$ from [11]. Intuitively, $\mathcal{F}_{\mathsf{tsig}}$ may be viewed as a simple repository of signed messages, and, crucially for

[14] Later on, for additional functionality, we define signing oracles that support arbitrary queries of their state.

the security analysis, $\mathcal{F}_{\mathsf{tsig}}$ does *not* hold any internal secrets. In more detail, $\mathcal{F}_{\mathsf{tsig}}$ operates according to the following specifications.

> **Activation & Signature request.** When activated, $\mathcal{F}_{\mathsf{tsig}}$ requests a verification algorithm $\mathcal{V}$ from the ideal adversary $\mathcal{S}$ (resulting from some externally calculated public key). Whenever a signature is requested by the parties for $m \in \{0,1\}^*$, $\mathcal{F}_{\mathsf{tsig}}$ keeps record of m and marks it as "signed".
>
> **Signature Verification.** When a message-signature pair (m, σ) is presented for verification, *if m is marked as "signed"*, then the functionality returns $\mathcal{V}(m, \sigma) \in \{\texttt{true}, \texttt{false}\}$, i.e. the output of the verification algorithm. Else, if m is not marked as "signed", $\mathcal{F}_{\mathsf{tsig}}$ returns `false`, regardless of $\mathcal{V}$.

So, in a nutshell, $\mathcal{F}_{\mathsf{tsig}}$ keeps a record of all messages that the parties agreed to sign, and it will outright reject any signature for any message that was not submitted for signing, including signature-strings that verify according to $\mathcal{V}$. As such, any protocol that realizes $\mathcal{F}_{\mathsf{tsig}}$ is a "good" threshold-signatures protocol.

Next, we describe sufficient conditions for a protocol to UC-realize $\mathcal{F}_{\mathsf{tsig}}$ in relation to unforgeability (which is a non-MPC security notion).

Definition (Simulatability – Informal)**.** Let $\mathcal{A}$ denote an adversary corrupting a subset of signatories in an execution of Σ. We say that Σ is $\mathcal{G}$-*simulatable*, if, for every $\mathcal{A}$, for every not-too-small ε, there exists $\mathcal{S}$ such that, with distinguishing advantage at most ε, $\mathcal{S}$ can simulate the interaction between $\mathcal{A}$ and the honest parties using the signing oracle $\mathcal{G}$. Notationally, letting $\mathrm{VIEW}_{\mathcal{A},\Sigma}$ and $\mathrm{OUT}_{\mathcal{S},\mathcal{G}}$ denote (i) $\mathcal{A}$'s view in an execution of Σ, and (2) $\mathcal{S}$'s output when interacting with $\mathcal{G}$ and $\mathcal{A}$, we write:

$$\forall \mathcal{A} \forall \varepsilon \; \exists S \quad \text{s.t. } \mathrm{VIEW}_{\mathcal{A},\Sigma} \stackrel{\varepsilon}{\equiv} \mathrm{OUT}_{\mathcal{S},\mathcal{G}}. \tag{3}$$

Remark 1.3. Simulatability is very weak compared to UC security, e.g. the simulator may depend on the code of the adversary (vs. a single $\mathcal{S}$ for all $\mathcal{Z}$ in the UC definition). In fact, simulatability is weaker than so-called "standalone" non-UC security (where $\mathcal{Z}$ does not interact with the experiments and simply outputs its view), because, as Equation (3) suggests, the running time of the simulator may depend on the distinguishing advantage ε.

Theorem (Unf. & Sim. → UC)**.** *If* SGN *is $\mathcal{G}$-Unforgeable and Σ is $\mathcal{G}$-Simulatable, then Σ UC-realizes* $\mathcal{F}_{\mathsf{tsig}}$*.*

Proof Sketch. Let $\mathcal{A}$ denote the real-world adversary and write $\mathcal{Z}$ for the environment. We begin by describing the ideal adversary $\mathcal{S}$ for Equation (2) , and it is assumed that $\mathcal{S}$ has black-box access to $\mathcal{A}$. First, $\mathcal{S}$ samples secrets for the honest parties, as prescribed by Σ. Second, to simulate the interaction of the honest parties with $\mathcal{A}$, $\mathcal{S}$ simply runs the code of the honest parties as prescribed by Σ. Third, for interacting with the ideal functionality, $\mathcal{S}$ submits $\mathcal{V}$ resulting from the public key of the interaction with $\mathcal{A}$ (as prescribed by Σ). Finally, $\mathcal{S}$ interacts with $\mathcal{Z}$ by simply relaying message between $\mathcal{A}$ and $\mathcal{Z}$.

Notice that $\mathcal{S}$ is *trivial* insofar as it simply runs the honest parties' code against the $\mathcal{A}$ (exactly like the real experiment) and it acts as a relaying vehicle between $\mathcal{A}$ and $\mathcal{Z}$. Therefore, $\mathcal{Z}$'s interaction with $\mathcal{A}$ in the real experiment and $\mathcal{Z}$'s interaction with $\mathcal{S}$ in the ideal experiment are identically distributed, in a *perfect* sense. It follows that the only way for $\mathcal{Z}$ to distinguish between real and ideal experiments is to *forge signatures*. Here is why: Assume that $\mathcal{Z}$ forges a signature $\sigma^\star$ for a message $m^\star$ that was never signed before. Then, $\mathcal{Z}$ asks a party $\mathcal{X}$ to verify the validity of the string $(m^\star, \sigma^\star)$, and $\mathcal{X}$ proceeds as follows. In the real world, $\mathcal{X}$ runs the verification algorithm on $(m^\star, \sigma^\star)$ and returns `true` (because the forgery verifies according to $\mathcal{V}$). On the other hand, in the ideal experiment, $\mathcal{X}$ queries $\mathcal{F}_{\mathsf{tsig}}$ which returns `false` because $m^\star$ was never submitted for signing, thus allowing $\mathcal{Z}$ to distinguish.

So, viewing $\mathcal{A}, \mathcal{Z}$ and $\mathcal{X}$ as a single PPTM, if $\mathcal{Z}$ distinguishes with noticeable probability α, then there exists PPTM $\mathcal{A}_0$ controlling a subset of parties in Σ that outputs a forgery with probability α at the end of the execution. By $\mathcal{G}$-simulatability, fixing $\varepsilon \leq \alpha/2$, there exists $\mathcal{S}_0$ that can simulate Σ in the presence of $\mathcal{A}_0$ with distinguishing advantage at most ε. Therefore, since $\varepsilon \leq \alpha/2$, it follows that $\mathcal{S}_0$'s output contains a forgery with probability at least $\alpha/2$, and so $\mathcal{G}$ is useful for forging signatures of SGN, in contradiction with the $\mathcal{G}$-unforgeability assumption.

In conclusion, no such $\mathcal{A}_0$ exists and Σ UC-realizes $\mathcal{F}_{\mathsf{tsig}}$. □

Remark 1.4. In a subsequent work, Makriyannis [33] extends the above to so-called *oracle-aided* signatures where each signature-string may depend on the query-answer pairs of a random oracle. As a corollary, by modeling the underlying hash function of the Schnorr signature scheme [39] as a random oracle, [33] shows that the simplest known protocol for Schnorr signatures [36, 38, 40], dubbed *Classic Schnorr* in [33], UC-realizes $\mathcal{F}_{\mathsf{tsig}}$. We emphasize that, herein, we make no assumptions about the underlying hash function of ECDSA, i.e. we do *not* model it as a random oracle.

Remark 1.5. Note that the above theorem and [33] have interesting ramifications for all non-pathological threshold-signatures protocols, i.e. protocols which are not essentially "broken". Namely, if a protocol Σ is not "broken" (either because it standalone/UC-realizes the natural functionality $\mathcal{F}_{\mathsf{SGN}}$, or it is unforgeable according to a suitable definition, e.g. simulatability), then Σ UC-realizes $\mathcal{F}_{\mathsf{tsig}}$. Thus, our theorem yields an equivalence between the game-based definition of unforgeability and the UC definition for realizing $\mathcal{F}_{\mathsf{tsig}}$.[15]

Finally, we note that Σ_{ecdsa} supports arbitrary concurrent signatures, i.e. there is no restriction on the number of concurrent signatures (or pre-signatures) being generated. So, by showing that our protocol Σ_{ecdsa} is simulatable (cf. Section 4.2) against adaptive adversaries, we deduce the following theorem.

Theorem (Security of Σ_{ecdsa} – Informal)**.** *Under suitable cryptographic assumptions, it holds that* Σ_{ecdsa} *UC-realizes* $\mathcal{F}_{\mathsf{tsig}}$ *with arbitrary concurrent signatures (or pre-signatures) against adaptive adversaries.*

2 Preliminaries

Hereafter, we write *presign*, *presigning* and *presignature* instead pre-sign, pre-signing and pre-signature.

2.1 Notation

Throughout the paper $\mathbb{Q}$, $\mathbb{Z}$ and $\mathbb{N}$ denote the set of rational, integer and natural numbers, respectively. Secret values are always denoted with lower case letters $(p, q, \ldots)$ and public values are *usually* denoted with upper case letters $(A, B, N, \ldots)$. 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 $\boldsymbol{X}$. Arrow-accented letters $\vec{A}, \vec{\rho}, \ldots$ denote ordered sets, i.e. tuples. Upper case bold letters $\boldsymbol{u}, \boldsymbol{v}$.. 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 $a \not| b$ for the negation. We write $\gcd : \mathbb{N}^2 \to \mathbb{N}$ for the gretest common divisor operation, $[a]_q$ denotes the modular reduction operation $a \mod q$, and $\varphi(\cdot)$ denotes Euler's totient function (not to be confused with ϕ which denotes a group homomorphism).

Groups & Fields. $\mathbb{G}, \mathbb{H}, \mathbb{K}$ denote groups and $\mathbb{F}$ is a field (typically we write $\mathbb{F}_q$ to specify that the field has q elements). We write $\mathbb{1} \in \mathbb{G}$ (or $\mathbb{H}$ or $\mathbb{K}$) for the identity element in $\mathbb{G}$ (or $\mathbb{H}$ or $\mathbb{K}$). Typically, $(\mathbb{G}, g, q)$ will denote the group-generator-order tuple for ECDSA. Group products, e.g. $\mathbb{K} = \mathbb{G}_1 \times \ldots \times \mathbb{G}_n$, are endowed with the natural group-product operation i.e. $\vec{A} \cdot \vec{B} = (A_i)_{i=1}^n \cdot (B_i)_{i=1}^n = (A_i *_i B_i)_{i=1}^n$, where $*_i$ is the group operation of $\mathbb{G}_i$. For $t \in \mathbb{Z}_N$, we write $\langle t \rangle = \{t^k \mod N \text{ s.t. } k \in \mathbb{Z}\}$ for the multiplicative group generated by t. For $\vec{e}_1, \ldots, \vec{e}_n \in \mathbb{F}^m$, where $\mathbb{F}$ is a field, write $\langle \vec{e}_1, \ldots, \vec{e}_n \rangle$ for the vector space generated by $\{\vec{e}_i\}_{i=1}^m$. For $\ell \in \mathbb{Z}$, we let $\pm\ell$ denote the interval of integers $\{-|\ell|, \ldots, 0, \ldots, |\ell|\}$.

Algorithms, Polynomials & Negligible Functions. We use sans-serif letters (enc, dec, $\ldots$) or calligraphic $(\mathcal{S}, \mathcal{A}, \ldots)$ to denote algorithms. U 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.

[15] In particular, the protocols [14, 16, 20, 24] UC-realize $\mathcal{F}_{\mathsf{tsig}}$ with the following caveat: our equivalence merely preserves the game-based security guarantee, and it does not extend the security guarantee beyond composability (where different sessions of the protocol are identified with distinct public keys). To illustrate this point, if a protocol Σ is secure only against static adversaries, then Σ realizes $\mathcal{F}_{\mathsf{tsig}}$ only against static adversaries, or, if Σ is secure only when the signature-generation process is sequential (non-concurrent), then Σ realizes $\mathcal{F}_{\mathsf{tsig}}$ only for sequentially-generated signatures (though different sessions of the protocol for different public keys can be composed arbitrarily).

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 $|\Pr[\mathcal{D}(1^\kappa, \boldsymbol{u}_\kappa) = 1] - \Pr[\mathcal{D}(1^\kappa, \boldsymbol{v}_\kappa) = 1]| \leq \varepsilon(\kappa)$ for every efficient distinguisher $\mathcal{D}$, 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.

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

2.2 Signatures and Unforgeability

Definition 2.1 (Signature Scheme.)**.** $\mathsf{SGN} = (\mathsf{gen}, \mathsf{sign}, \mathsf{vrfy})$ is a threetuple of algorithms such that

1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{gen}(1^\kappa)$, where κ is the security parameter.
2. For $\mathrm{msg} \in \{0,1\}^*$, $\sigma \leftarrow \mathsf{sign}_{\mathsf{sk}}(\mathrm{msg})$.
3. For $\mathrm{msg}, \sigma \in \{0,1\}^*$, $\mathsf{vrfy}_{\mathsf{pk}}(\sigma, \mathrm{msg}) = b \in \{0,1\}$.

   Correctness. For $\sigma \leftarrow \mathsf{sign}_{\mathsf{sk}}(\mathrm{msg})$, it holds that $\mathsf{vrfy}_{\mathsf{pk}}(\sigma, \mathrm{msg}) = 1$.

Existential Unforgeability. Next, we define security for signature schemes.

FIGURE 3 (Augmented signature oracle $\mathcal{G}$)

Parameters. Signature scheme SGN and randomized functionality f.

Operation.

1. On input $(\mathsf{gen}, 1^\kappa)$, generate a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{gen}(1^\kappa)$, initialize $\mathtt{state} = (\mathsf{sk}, \mathsf{pk})$, and return pk.
   Ignore future calls to gen.
2. On input x, sample $r \leftarrow \$$ and return $\tau = f(x, \mathtt{state}; r)$.
   Update $\mathtt{state} := \mathtt{state} \cup \{(x, \tau; r)\}$.

Figure 3: Augmented signature oracle $\mathcal{G}$

FIGURE 4 ($\mathcal{G}$-Existential Unforgeability Experiment $\mathcal{G}\text{-}\mathsf{EU}(\mathcal{A}, 1^\kappa)$)

1. Call $\mathcal{G}$ on $(\mathsf{gen}, 1^\kappa)$ and hand pk to $\mathcal{A}$.
2. The adversary $\mathcal{A}$ makes $n(\kappa)$ adaptive calls to $\mathcal{G}$ for $n \in \mathsf{poly}$.
3. $\mathcal{A}$ outputs (m, σ) given its view (randomness and query-answer pairs to $\mathcal{G}$)

- **Output:** $\mathcal{G}\text{-}\mathsf{EU}(\mathcal{A}, 1^\kappa) = 1$ if $\mathsf{vrfy}_{\mathsf{pk}}(m, \sigma) = 1$ and m was not queried by $\mathcal{A}$ when calling $\mathcal{G}$.

Figure 4: $\mathcal{G}$-Existential Unforgeability Experiment $\mathcal{G}\text{-}\mathsf{EU}(\mathcal{A}, 1^\kappa)$

Definition 2.2 ($\mathcal{G}$-Existential Unforgeability.)**.** We say that SGN is $\mathcal{G}$-existentially unforgeable if for all $\mathcal{A}$ there exists $\nu \in \mathsf{negl}$ s.t. $\Pr[\mathcal{G}\text{-}\mathsf{EU}(\mathcal{A}, 1^\kappa) = 1] \leq \nu(\kappa)$, where $\mathcal{G}\text{-}\mathsf{EU}(\cdot)$ denotes the security game from Figure 4.

2.3 MPC and Universal Composability

We use the simplified variant of the UC framework (which is sufficient for our purposes because the identities of all parties are assumed to be fixed in advance). In this section we provide a quick reminder of the framework.

The model for n-party protocol Π. For the purpose of modeling the protocols in this work, we consider a system that consists of the following $n+2$ *machines,* where each machine is a computing element (say, an interactive Turing machine) with a specified program and and identity. First, we have n machines with program Π and identities $\mathcal{P}_1, \ldots, \mathcal{P}_n$. Next, we have a machine $\mathcal{A}$ representing the adversary an a machine $\mathcal{Z}$ representing the environment. All machines are initialized on a security parameter κ and are polynomial in κ. The environment $\mathcal{Z}$ is activated first, with an external input z. $\mathcal{Z}$ activates the parties, chooses their input and reads their output. $\mathcal{A}$ can corrupt parties and instruct them to leak information to $\mathcal{A}$ and to perform arbitrary instructions. $\mathcal{Z}$ and $\mathcal{A}$ communicate freely throughout the computation. The real process terminates when the environment terminates. Let $\mathrm{EXEC}^{\mathcal{Z}}_{\Pi,\mathcal{A}}(1^\kappa, z)$ denote the environment's output in the above process.

We assume for simplicity that the parties are connected via an authenticated, synchronous broadcast channel. That is, the computation proceeds in rounds, and each message sent by any of the parties at some round is made available to all parties at the next round. Formally, synchronous communication is modeled within the UC framework by way of $\mathcal{F}_{\text{syn}}$, the ideal synchronous communication functionality from [6, Section 7.3.3]. The broadcast property is modeled by having $\mathcal{F}_{\text{syn}}$ require that all messages are addressed at all parties.

Ideal Process. the ideal process is identical to the real process, with the exception that now the machines $\mathcal{P}_1, \ldots, \mathcal{P}_n$ do not run Π, Instead, they all forward all their inputs to a subroutine machine, called the *ideal functionality* $\mathcal{F}$. Functionality $\mathcal{F}$ then processes all the inputs locally and returns outputs to $\mathcal{P}_1, \ldots, \mathcal{P}_n$. Let $\mathrm{EXEC}^{\mathcal{Z}}_{\mathcal{F},\mathcal{S}}(1^\kappa, z)$ denote the environment's output in the above process.

Definition 2.3. We say that Π UC-realizes $\mathcal{F}$ if for every adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that for every environment $\mathcal{Z}$ there exists $\nu \in \mathsf{negl}$ such that

$$\{\mathrm{EXEC}^{\mathcal{Z}}_{\Pi,\mathcal{A}}(1^\kappa, z)\}_{z\in\{0,1\}^*} \stackrel{\nu}{\equiv} \{\mathrm{EXEC}^{\mathcal{Z}}_{\mathcal{F},\mathcal{S}}(1^\kappa, z)\}_{z\in\{0,1\}^*}$$

The Adversarial Model. The adversary can corrupt parties adaptively throughout the computation. Once corrupted, the party reports all its internal state to the adversary, and from now on follows the instructions of the adversary. We also allow the adversary to *leave*, or *decorrupt* parties. A decorrupted party resumes executing the original protocol and is no longer reporting its state to the adversary. Still, the adversary knows the full internal state of the decorrupted party at the moment of decorruption. Finally, the real adversary is assumed to be *rushing*, i.e. it receives the honest parties' messages before it sends messages on behalf of the corrupted parties.

Global Functionalities. It is possible to capture UC with global functionalities within the plain UC framework. Specifically, having Π UC-realize ideal functionality $\mathcal{F}$ in the presence of global functionality $\mathcal{G}$ is represented by having the protocol $[\Pi, \mathcal{G}]$ UC-realize the protocol $[\mathcal{F}, \mathcal{G}]$ within the plain UC framework. Here $[\Pi, \mathcal{G}]$ is the $n+1$-party protocol where machines $\mathcal{P}_1, \ldots, \mathcal{P}_n$ run Π, and the remaining machine runs $\mathcal{G}$. Protocol $[\mathcal{F}, \mathcal{G}]$ is defined analogously, namely it is the $n+2$-party protocol where the first $n+1$ machines execute the ideal protocol for $\mathcal{F}$, and the remaining machine runs $\mathcal{G}$.

2.3.1 Proactive Threshold Signatures

Definition 2.4 (Proactive Threshold Signatures)**.** Let $\Sigma = (\Sigma_{\mathsf{kgen}}, \Sigma_{\mathsf{refr}}, \Sigma_{\mathsf{pres}}, \Sigma_{\mathsf{sign}})$ denote a protocol for parties in $\boldsymbol{P} = \{\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_n\}$ parametrized by $\mathsf{QRMs} \subseteq 2^{\boldsymbol{P}}$. We say that Σ is a proactive Threshold-Signatures scheme for $\mathsf{SGN} = (\ldots, \mathsf{vrfy})$ if it offers the following functionality.

1. Σ_{kgen} takes input 1^κ from $\mathcal{P}_i \in \boldsymbol{P}$ and returns (pk, s_i) to each $\mathcal{P}_i \in \boldsymbol{P}$.
2. Σ_{refr} takes input (pk, s_i) from each $\mathcal{P}_i \in \boldsymbol{P}$ and returns (a fresh) value s_i to each $\mathcal{P}_i \in \boldsymbol{P}$.
3. Σ_{pres} takes input $(\mathsf{pk}, s_i, \boldsymbol{Q}, L)$ from each $\mathcal{P}_i \in \boldsymbol{Q}$ and returns $(w_{i,1}, \ldots, w_{i,L})$ to each $\mathcal{P}_i$.
4. Σ_{sign} takes input $\mathrm{msg} \in \{0,1\}^*$ and $(\mathsf{pk}, s_i, \boldsymbol{Q}, \ell)$ from $\mathcal{P}_i \in \boldsymbol{Q}$ and returns σ to (at least one) $\mathcal{P}_i$.

Correctness. Using the notation above, if $\boldsymbol{Q} \in \mathsf{QRMs}$ then $\mathsf{vrfy}_{\mathsf{pk}}(\sigma, \mathrm{msg}) = 1$ in an honest execution.

Sets $\boldsymbol{Q} \in \mathsf{QRMs}$ are called *quorums* and the span between two consecutive executions of Σ_{refr} is referred to as an *epoch*. By convention, the span before the first execution of Σ_{refr} is the first epoch.

A protocol Σ is said to be secure if it UC-realizes functionality $\mathcal{F}_{\text{tsig}}$, defined below.

2.3.2 Ideal Threshold-Signatures Functionality

We use the ideal functionality $\mathcal{F}_{\mathsf{tsig}}$ of [11], which generalizes the non-threshold signature functionality of Canetti [7]. We briefly outline $\mathcal{F}_{\mathsf{tsig}}$ next and we refer the reader to the appendix (p. 41) for the full description.

For each signing request for a message msg, the functionality requests a signature string σ from the adversary, which is submitted from the outside, i.e. the signature string σ is not calculated internally from the ideal functionality. Once sigma is submitted by the adversary, the functionality keeps record of (msg, σ). When a party submits a pair (msg', σ') for verification, the functionality simply returns *true* if it has record of that pair and *false* otherwise.

For proactive security, the functionality admits an additional interface for recording corrupted and decorrupted parties. When a party is decorrupted, the functionality records that party as *quarantined* until it is instructed to purge that record (via a special key-refresh interface). If all parties are corrupted and/or quarantined at any given time, then the functionality enters a pathological mode of operation and it ignores the message-signature repository it holds internally.

2.3.3 Global Random Oracle

We follow formalism of [5, 8] for incorporating the random oracle into the UC framework. In particular, we use the *strict global random oracle* paradigm which is the most restrictive way of defining a random oracle, defined below.

FIGURE 5 (The Global Random Oracle Functionality $\mathcal{H}$)

Parameter: Output length h.

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

Figure 5: The Global Random Oracle Functionality $\mathcal{H}$

2.4 Group/Number-Theoretic Definitions

Definition 2.5. We say that $N \in \mathbb{N}$ is a biprime if N admits exactly two non-trivial prime divisors. In particular, a biprime N is a Paillier-Blum integer iff $\gcd(N, \varphi(N)) = 1$ and $N = pq$ for primes $p, q \equiv 3 \mod 4$.

Definition 2.6 (Paillier Encryption)**.** Define the Paillier cryptosystem as the three tuple $(\mathsf{gen}, \mathsf{enc}, \mathsf{dec})$ below.

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

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

Definition 2.7 (ECDSA)**.** Let $(\mathbb{G}, g, q)$ denote the group-generator-order tuple associated with a given curve. We recall that elements in $\mathbb{G}$ are represented as pairs $a = (a_x, a_y)$, where the a_x and a_y are referred to as the projection of a on the x-axis and y-axis respectively, denoted $a_x = a|_{x\text{-axis}}$ and $a_y = a|_{y\text{-axis}}$, respectively. The security parameter below is implicitly set to $\kappa = \log(q)$.

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

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

Definition 2.8 (El-Gamal Commitment)**.** For group-generator-order tuple $(\mathbb{G}, g, q)$ define algorithm com which takes input $(k, Y) \in \mathbb{F}_q \times \mathbb{G}$ and returns $(\vec{B}; \beta)$ such that $\vec{B} = (g^\beta, Y^\beta g^k)$ for randomizer $\beta \leftarrow \mathbb{F}_q$.

2.5 Schnorr Protocols

Let κ denote the security parameter. Hereafter, let $\phi : \mathbb{H} \to \mathbb{G}$ denote a group homomorphism from $(\mathbb{H}, +)$ to $(\mathbb{G}, \cdot)$ and $\boldsymbol{E} \subseteq \mathbb{Z}$, and $\boldsymbol{R}, \boldsymbol{S} \subseteq \mathbb{H}$. It is assumed that (the description of) the tuple $(\phi, \mathbb{H}, \mathbb{G}, \boldsymbol{E}, \boldsymbol{R}, \boldsymbol{S})$ is efficiently generated by a PPTM with input κ, and ϕ is efficiently computable as a function of κ.

Definition 2.9. An *m-batch* Schnorr protocol Π for tuple $(\phi, \boldsymbol{E}, \boldsymbol{R}, \boldsymbol{S})$ consists of the following interactive process. For common input $(X_i)_{i=1}^m \in \mathbb{G}^m$ and secret input $(w_i)_{i=1}^m \in \mathbb{H}^m$:

1. Prover samples $\alpha \leftarrow \boldsymbol{S}$ and sends $A = \phi(\alpha)$ to the verifier.
2. Verifier replies with $\vec{e} = (e_1 \ldots e_n) \leftarrow \boldsymbol{E}^m$.
3. Prover sends $z = \alpha + \sum_{j=1}^m e_j \cdot w_j \in \mathbb{H}$, where $e \cdot w = \underbrace{w_j^* + \ldots + w_j^*}_{|e| \text{ times}}$ and $w_j^* = \begin{cases} w_j & \text{if } e \geq 0 \\ -w_j & \text{otherwise} \end{cases}$.

   Check: Verifier accepts if and only if $\phi(z) = A \cdot \prod_{j=1}^m X_j^{e_j} \in \mathbb{G}$ and $z \in \boldsymbol{S}$.

If $\boldsymbol{R}$ and $\boldsymbol{S}$ are not specified, then it is assumed that $\boldsymbol{R} = \boldsymbol{S} = \mathbb{H}$. The protocol is (μ, ν)-secure if it satisfies

μ-HVZK. If $X_i = \phi(w_i)$ and $w_i \in \boldsymbol{R}$ for all i, then $\tau = (\vec{X}, A, \vec{e}, z)$ for $z \leftarrow \boldsymbol{S}$, $\vec{e} \leftarrow \boldsymbol{E}^m$ and $A = \phi(z) \cdot \prod_{j=1}^m X_j^{-e_j}$ is statistically μ-close to an honest transcript.

ν-Soundness. If $\exists j$ such that $\phi(w_j) \neq X_j$, for every $w \in \boldsymbol{S}$, then, for every $A \in \mathbb{G}$ in Item 1, the probability that the verifier accepts is at most ν.

Further define

$(\varepsilon, \boldsymbol{V})$-Extractibility. For all efficient $\mathcal{A}$, if $\{\tau_j = (\vec{X}, A, \vec{e}_j, z_j)\}_{j=1}^{m+1} \leftarrow \mathcal{A}$ are $m+1$ valid transcripts for common input $\{X_j\}_{j=1}^{m+1}$ such that $\{\vec{e}_1, \ldots, \vec{e}_{m+1}\} \in \boldsymbol{V}$, then, with all but probability ε, there exists an efficient PPTM $\mathcal{E}$ such that $A = \phi(\alpha)$ and $X_j = \phi(w_j)$ for $\alpha, \{w_j\}_j \leftarrow \mathcal{E}(\{X_j, \tau_j\}_{j=1}^{m+1})$.

2.5.1 Embedded Schnorr Protocols

Unfortunately, many tuples $(\phi, \boldsymbol{E}, \boldsymbol{R}, \boldsymbol{S})$ of interest do not give rise to sound Schnorr protocols. To guarantee soundness, we embed the desired homomorphism ϕ into a larger one $\hat{\phi}$. Namely, let $(\phi, \boldsymbol{E}, \boldsymbol{R}, \boldsymbol{S})$ and $(\hat{\phi}, \boldsymbol{E}, \hat{\boldsymbol{R}}, \hat{\boldsymbol{S}})$ for $\phi : \mathbb{H} \to \mathbb{G}$ and $\hat{\phi} : \hat{\mathbb{H}} \to \hat{\mathbb{G}}$ such that

- $\hat{\mathbb{H}} = \mathbb{H} \times \mathbb{K}$ and $\hat{\phi}(u, v) = (\phi(u), \theta(u, v))$.
- $(u, v) \in \hat{\boldsymbol{R}}$ and $(u, v) \in \hat{\boldsymbol{S}}$ implies $u \in \boldsymbol{R}$ and $u \in \boldsymbol{S}$ respectively.

Definition 2.10. Define the m-batch *embedded* Schnorr protocol for tuples $(\hat{\phi}, \boldsymbol{E}, \hat{\boldsymbol{R}}, \hat{\boldsymbol{S}})$ and $(\phi, \boldsymbol{E}, \boldsymbol{R}, \boldsymbol{S})$ to consist of the following interactive process. For common input $\vec{X} \in \mathbb{G}^m$ and secret input $\vec{w} \in \mathbb{H}^m$:

1. Prover samples $(\vec{u}, \vec{v}) \leftarrow \hat{\boldsymbol{R}}^m|_{\vec{u}=\vec{w}}$ and $\alpha \leftarrow \hat{\boldsymbol{S}}$ and sends $(A, \vec{Y}) = (\hat{\phi}(\alpha), \theta(u_1, v_1), \ldots)$.
2. Verifier replies with $\vec{e} \leftarrow \boldsymbol{E}^m$.

3. Prover sends $z = \alpha + \sum_{i=1}^{m} e_i \cdot (u_i, v_i) \in \mathbb{H}$ to the verifier.

   Check: Verifier sets $\hat{X}_i = (X_i, Y_i)$ and accepts iff $\hat{\phi}(z) = A \cdot \prod_{i=1}^{m} \hat{X}_i^{e_i} \in \mathbb{G}$ and $z_i \in \hat{\boldsymbol{S}}$, for all i.

Soundness and *extractability* are defined analogously to non-embedded protocols. For HVZK:

> *μ-HVZK.* There exists an efficient sampler $\mathcal{I}$ for the extended input $\vec{Y} \in \mathbb{K}^m$ such that if $X_i = \phi(w_i)$ and $w_i \in \boldsymbol{R}$ for all i, then $\tau = (\vec{X}, \vec{Y}, A, \vec{e}, z)$ for $\vec{Y} \leftarrow \mathcal{I}(X), z \leftarrow \hat{\boldsymbol{S}}$, $\vec{e} \leftarrow \boldsymbol{E}^m$ and $A = \phi(z) \cdot \prod_{j=1}^{m} (X_j, Y_j)^{-e_j}$ is statistically μ-close to an honest transcript.

2.5.2 NIZK and the Fiat-Shamir Transform

We make extensive use of the Fiat-Shamir transform for converting an interactive zero-knowledge protocol into an non-interactive zero-knowledge proof. Namely, consider the process from Figure 6. (The process is analogous for embedded Schnorr protocols)[16]

FIGURE 6 (Schnorr Proof in ROM $\psi \leftarrow \Pi^{\mathsf{FS}}(\mathsf{aux}, \vec{X}; \vec{w})$)

Parameters: Schnorr protocol Π for tuple $(\phi, \boldsymbol{E}, \boldsymbol{R}, \boldsymbol{S})$ and random oracle $\mathcal{H}$.

1. Sample $\alpha \leftarrow \boldsymbol{S}$ and set $A = \phi(\alpha)$.
2. Calculate $\vec{e} = (e_1, \ldots, e_m) = \mathcal{H}(\mathsf{aux}, \vec{X}, A)$.

   Output: $\psi = (A, \vec{e}, z)$ for $z = \alpha + \sum_{j=1}^{m} e_j w_j \in \mathbb{H}$

Figure 6: Schnorr Proof in ROM $\psi \leftarrow \Pi^{\mathsf{FS}}(\mathsf{aux}, \vec{X}; \vec{w})$

Notation 2.11. As shown above, for a m-batch Schnorr protocol Π for tuple $(\phi, \boldsymbol{E}, \boldsymbol{R}, \boldsymbol{S})$, we write Π^{FS} for the non-interactive process resulting by applying the Fiat-Shamir transform to Π. Furthermore, we write $\psi \leftarrow \Pi^{\mathsf{FS}}(\mathsf{aux}, \vec{X}; \vec{w})$ for the output of the process on common (public) input $(\mathsf{aux}, \vec{X})$ and secret input $\vec{w}$.

Definition 2.12. We say that $\psi = (A, \vec{e}, z)$ is a valid *proof* for Π^{FS} on input (aux, X) if $\phi(z) = A \cdot \prod_k X_k^{e_k}$ for $\vec{e} = \mathcal{H}(\mathsf{aux}, X, A)$ and $z \in \boldsymbol{S}$. Furthermore, we say that Π^{FS} is a secure proof system in the ROM if

> *Zero-Knowledge.* If $X_i = \phi(w_i)$ and $w_i \in \boldsymbol{R}$ for all i, then $\tau = (\vec{X}, A, \vec{e}, z)$ for $z \leftarrow \boldsymbol{S}$, $\vec{e} \leftarrow \boldsymbol{E}^m$ and $A = \phi(z) \cdot \prod_{j=1}^{m} X_j^{-e_j}$ is statistically close to $\psi \leftarrow \Pi(\vec{X}, \mathsf{aux}; \vec{w})$, for every $\mathsf{aux} \in \{0,1\}^*$.
>
> *Soundness.* If $\exists j$ such that $\phi(w) \neq X_j$, for every $w \in \boldsymbol{S}$, then the probability that an efficient PPTM outputs a valid proof is negligible.

Fact 2.13. *If Π is (μ, ν)-secure for $\mu, \nu \in \mathsf{negl}(\kappa)$, then Π^{FS} is a secure proof system in the ROM.*

Remark 2.14 (Optimization in ROM). It is possible to reduce the communication complexity of Π^{FS} such that $\hat{\psi} = (u, z)$ where $u = \mathcal{H}(\mathsf{aux}, A = \phi(A))$, using the notation from Figure 6. Accordingly, the verifier accepts $\hat{\psi}$ if $u = \mathcal{H}(\mathsf{aux}, g^z \cdot \prod_{\ell=1}^{m} X^{-e_\ell})$ for $\vec{e} = \mathcal{H}(\mathsf{aux}, \vec{X}, u)$. Note that this optimization is mostly relevant for proofs that cannot be batched (e.g. $\psi_{j,\ell} \leftarrow \Theta_{R_j}^{\mathsf{FS}}$ from Definition 3.7).

2.5.3 Proof Aggregation in ROM

Let Π denote (μ, ν)-secure Schnorr protocol for $(\phi, \boldsymbol{E}, \boldsymbol{R}, \boldsymbol{S})$ and consider the proof-aggregation process from Figure 7 among n provers $\mathcal{P}_1, \ldots, \mathcal{P}_n$. In this section, we show that the properties of soundness and zero-knowledge are preserved for the aggregated variant of the protocol. Regarding soundness, clearly, even if all the parties collude, by Fact 2.13 it is not feasible to generate a valid proof ψ if the common input $\vec{X}$ does not admit a suitable preimage. For zero-knowledge, we require the following notions.

Let $\mathcal{A}$ denote an adversary corrupting a strict subset of the $\mathcal{P}_i$'s in Figure 7 and write $\mathrm{Real}_{\mathcal{A}}^{\Pi}(1^\kappa, u, v)$ for the adversary's view in an execution of the protocol with common input $u = (\mathsf{aux}, \vec{X})$ and auxiliary input

[16]Do not to forget to hash the extended input $\vec{Y}$.

Party $\mathcal{P}_i$ — Party $\mathcal{P}_j$

$\alpha_i \leftarrow \mathbb{H}$ and set $A_i = \phi(\alpha_i)$

Set $C_i = \mathcal{H}(\mathsf{aux}, \mathcal{P}_i, \vec{X}_i, A_i, \rho_i)$ for $\rho_i \leftarrow \{0,1\}^\kappa$

$C_i \rightarrow$ / $\leftarrow C_j$

When obtaining all $(C_j)_{j \neq i}$

$A_i, \vec{X}_i, \rho_i \rightarrow$ / $\leftarrow A_j, \vec{X}_j, \rho_j$

$\forall j$, check $C_j = \mathcal{H}(\mathsf{aux}, \mathcal{P}_j, \vec{X}_j, A_j, \rho_j)$

Set $A = \prod_{k=1}^n A_k$ and calculate

$\vec{e} = \mathcal{H}(\mathsf{aux}, \vec{X}, A)$ and $z_i = \alpha_i + \sum_\ell e_\ell w_{i,\ell}$

$z_i \rightarrow$ / $\leftarrow z_j$

Output $(\psi, \vec{X})$ where $\vec{X} = \prod_\ell \vec{X}_\ell$ and $\psi = (A, \vec{e}, z)$

if $\phi(z) = A \cdot \prod_\ell X_\ell^{e_\ell}$ and $z = \sum_{k=1}^n z_k \in \boldsymbol{S}$

Figure 7: NIZK Aggregation $\Pi^{\mathtt{AGT}}$ for common input $(\mathsf{aux}, \vec{X} = \prod_i \vec{X}_i)$

$v \in \{0,1\}^*$. It is assumed that $\mathcal{A}$ has oracle access to the random oracle $\mathcal{H}$ and $\vec{X}_j = (X_{j,1}, \ldots, X_{j,m})$ admits a suitable preimages $\{w_{j,\ell} \in \boldsymbol{R}\}_{\ell \in [m]}$ if $\mathcal{P}_j$ is not corrupted by $\mathcal{A}$. Write $\mathcal{S}$ for a PPTM with taking auxiliary input with blackbox access to $\mathcal{A}$ and oracle access to $\mathcal{H}$, and let $\mathrm{Ideal}_{\mathcal{S}}(1^\kappa, u, v)$ for the output of $\mathcal{S}$.

Definition 2.15 (ZK for Aggregation)**.** Using the notation above, we say that the Schnorr aggregation protocol $\Pi^{\mathtt{AGT}}$ is μ-ZK if for all $\mathcal{A}$, $u = (\mathsf{aux}, \vec{X})$ and $v \in \{0,1\}^*$, there exists $\mathcal{S}$ such that

$$\mathrm{SD}(\mathrm{Real}^{\Pi}_{\mathcal{A}}(1^\kappa, u, v), \mathrm{Ideal}_{\mathcal{S}}(1^\kappa, u, v)) \leq \mu.$$

Claim 2.16. *If Π is μ-HVZK then $\Pi^{\mathtt{AGT}}$ is $(n\mu)$-ZK.*

Proof. Write $\boldsymbol{H}$ for the parties not corrupted by $\mathcal{A}$. For all $\mathcal{P}_j \in \boldsymbol{H}$, run the HVZK simulator from Definition 2.9 to obtain $\psi_j = (A_j, \vec{e}, z_j)$ (notice that each ψ_j contains the same $\vec{e}$). Set $C_j = \mathcal{H}(\mathsf{aux}, \mathcal{P}_j, A_j, \rho_j)$ for $\rho_j \leftarrow \{0,1\}^\kappa$; hand over $\{C_j\}_{j \in \boldsymbol{H}}$ to $\mathcal{A}$. When obtaining C_i for all corrupted $\mathcal{P}_i$'s, retrieve $(\mathsf{aux}, \mathcal{P}_i, A_i, \rho_i)$ from $\mathcal{A}$'s oracle queries and set $A = \prod_{i=1}^k A_k$. Define $\mathcal{H}'$ such that $\mathcal{H}' \equiv \mathcal{H}$ except for $\mathcal{H}'(\mathsf{aux}, \vec{X}, A) = \vec{e}$; hereafter answer oracle queries according to $\mathcal{H}'$ (rather than $\mathcal{H}$). Conclude the simulation by handing over (A_j, ρ_j) and then z_j for every $\mathcal{P}_j \in \boldsymbol{H}$. The claim follows from the μ-HVZK property of the underlying zero-knowledge simulator, and union bound. □

3 Protocol

In this section we define our proactive threshold-ECDSA protocol $\Sigma_{\mathsf{ecdsa}} = (\Sigma_{\mathsf{kgen}}, \Sigma_{\mathsf{refr}}, \Sigma_{\mathsf{pres}}, \Sigma_{\mathsf{sign}})$. To keep the protocol description as simple as possible, we opted to ignore the key-refresh; so $\Sigma_{\mathsf{refr}} = \perp$ herein. However, key-refresh can be easily supported by essentially re-executing Σ_{kgen} (with appropriate checks), similarly to the protocols of [9, 11]. Each phase of the protocol is defined in Figure 8 (key generation Σ_{kgen}), Figure 9 (presigning Σ_{pres}) and Figure 11 (signing Σ_{sign}). Furthermore, we opted to present the protocol in its two-party variant with a single online party $\mathcal{P}_\infty$, and, to ease the transition to the multiparty case, we opted to describe the online party $\mathcal{P}_\infty$ *as if* it locally emulates (mocks) the cosignatories $\mathcal{P}_1, \ldots, \mathcal{P}_n$, and according to a running index $i \in [n]$. Thus, the multiparty version of our protocol follows straightforwardly by simply "parallelizing" $\mathcal{P}_\infty$ with respect to identifier $i \in [n]$.

Notation and Conventions. Recall that κ denotes the security parameter and fix the ZK parameter ν s.t. $\nu - 2\kappa \in \omega(\log(\kappa))$. For $M \in \mathbb{N}$, let $\boldsymbol{I}(M) = \pm M/2$ and $\boldsymbol{J}(M) = \pm M \cdot 2^{\nu-1}$. Let λ and m denote the packing parameter and batching parameter, respectively. Let τ denote the pack shift, i.e. $\sum_{j=1}^{\lambda} w_{j-1} 2^{(j-1)\tau}$ is a packing of the values $w_0, \ldots, w_\lambda$ for w_j such that $\log(w_j) < \tau$. The parties also hold a common input $\mathsf{aux} \in \{0,1\}^*$ that specifies the session identifier (sid) as well as the parties' identities (pid's). Furthermore, it is assumed that aux is provided as input to the random oracle when generating proofs and thus all proofs ψ are generated as $\psi \leftarrow \Pi^{\mathsf{FS}}(\mathsf{aux}, X; w)$ (we simply write $\Pi^{\mathsf{FS}}(X; w)$ to reduce clutter) where Π is a Schnorr protocol, and each protocol is described in the relevant subsection, when needed.

Parameter	Security/Soundness	ZK	Online Parties	Packing	Batching
Notation	κ	$\nu - 2\kappa$	n	λ	m
Running index	N/A	N/A	i	j	ℓ

Table 3: Summary of parameters/indices of the protocol

To avoid clutter and to improve readability, we opted to suppress the randomizers in the protocol descriptions, e.g. we write $C = \mathsf{enc}_i(k)$ for the encryption of k under the Paillier key of $\mathcal{P}_i$ and it is assumed that the randomizer is chosen according to the encryption process; the randomizers are crucial for the ZK-proofs, so in the description of the Schnorr protocols we do call attention to these values.

Furthermore, the protocol description does not explicitly mention proof verification. However, it goes without saying, every time $\mathcal{P}_i$ obtains a proof ψ, the protocol instructs $\mathcal{P}_i$ to verify ψ against the available data.

3.1 Pedersen Parameters

Before we turn to the protocol description, we introduce the Pedersen parameter, arguably the "secret sauce" for achieving security.

Definition 3.1 (Pedersen Parameters)**.** Define algorithm ped s.t. $\pi = (\hat{N}, t, s_1, \ldots, s_m, \psi) \leftarrow \mathsf{ped}(1^\kappa)$ where

1. $\hat{N} = pq$ of size $O(\kappa)$ such that $p = 2p' + 1$ and $q = 2q' + 1$ and p', q' are all prime.
2. $t \leftarrow \mathrm{QR}(\mathbb{Z}^*_{\hat{N}})$ and $s_1, \ldots, s_m \leftarrow \langle t \rangle$.
3. $\psi \leftarrow (\Pi^*)^{\mathsf{FS}}(s_1, \ldots, s_m)$ where Π^* is the m-batch Schnorr protocol for $(\phi, \boldsymbol{E})$ where $\boldsymbol{E} = \{0,1\}$ and $\phi(\alpha) = t^\alpha \mod \hat{N}$.

Claim 3.2. *It holds that* Π^* *is* (μ, ν)*-secure for* $\mu = 1 - \frac{\varphi(N)}{N}$ *and* $\nu = \frac{1}{2}$.

Proof. c.f. Claim B.1, Appendix B.1. (Note that soundness may be amplified via parallel repetition) □

Remark 3.3. We note that many Schnorr protocol herein arise as embedded Schnorr protocols (c.f. Definition 2.10) where the underlying homomorphism is augmented to incorporate π_i (the Pedersen parameter of a party $\mathcal{P}_i$). Looking ahead to the security analysis, note that the well-formedness of π is crucial for the soundness of resulting Schnorr protocol, and partial well-formedness of π (according to Item 3 above) is crucial for the HVZK property, cf. Claims 3.6 and 3.9.

Summary of Symbols. In Table 4, we provide a convenient summary for the different keys/params held by the parties. We recall that Pallier encryption and El-Gamal commitments are described in Definitions 2.6 and 2.8 respectively.

Parameter	ECDSA sk	ECDSA pk	El-Gamal pk	Paillier pk	Pedersen param
Notation	x	X	Y	N/A	N/A
$\mathcal{P}_i$'s Contribution /Value	x_i	X_i	Y_i	N_i	π_i

Table 4: Summary of keys/params held by the parties

3.2 Key-Generation & Presign

The key generation (c.f Figure 8) contains a ZK protocol, Φ_π, which is not fully expressible as a Schnorr protocol and we have we deferred the details of Φ_π to Appendix B.2.[17] For the remainder, it suffices to note that Φ_π yields the validity of the tuple (N, W, X) as described in Definition 3.4 below.

Definition 3.4. For fixed $(\mathbb{G}, g, q)$, define $\boldsymbol{A}$ to consist of all tuples $(N, W, X; p_1, p_2, x)$ such that (i) N is a Paillier-Blum modulus (c.f. Definition 2.5) with prime factors p_1, p_2 and $p_1, p_2 \geq q$, (ii) $\mathsf{dec}_{\varphi(N)}(W) = x$ and $X = g^x \in \mathbb{G}$ and (iii) $x \in [0, q-1]$. If W and X are not specified, assume that $(W, X) = (1, \mathbb{1})$ and $x = 0$.

3.2.1 ZK for Key-Generation & Presigning

Definition 3.5. For $(\hat{N}, t, s_1, \ldots) = \pi_i$, define Ξ_i to be the Schnorr protocol for $(\phi, \boldsymbol{E}, \boldsymbol{R}, \boldsymbol{S})$ where $\boldsymbol{E} = \boldsymbol{I}(2^\kappa)$,

$$\begin{aligned}\phi : \mathbb{Z}^\lambda \times \mathbb{F}_q^\lambda \times \mathbb{Z}_N^* \times \mathbb{Z} \to \mathbb{G}^{2\lambda} \times \mathbb{Z}_{N^2}^* \times \mathbb{Z}_{\hat{N}}^* \\ (\vec{w}, \vec{\mu}, v, \rho) \mapsto (\phi_1(\vec{w}, \vec{\mu}), \phi_2(\vec{w}, v), \phi_3(\vec{w}, \rho))\end{aligned}$$

and

$$\begin{cases} \phi_1 : (\vec{w}, \vec{\mu}) \mapsto ((g, Y)^{\mu_i} \cdot (\mathbb{1}, g)^{w_j})_{j=1}^\lambda \\ \phi_2 : (\vec{w}, v) \mapsto \prod_{j=1}^\lambda (1 + 2^{\tau \cdot (j-1)} \cdot N)^{w_j} \cdot v^N \mod N^2 \\ \phi_3 : (\vec{w}, \rho) \mapsto t^\rho \cdot \prod_{j=1}^\lambda s_j^{w_j} \mod \hat{N} \end{cases}$$

and

$$\begin{cases} (\vec{w}, \vec{\mu}, v, \rho) \in \boldsymbol{R} \iff \rho \in \boldsymbol{I}(\hat{N} \cdot 2^\kappa) \ \wedge\ \forall j\ w_j \in \boldsymbol{I}(2^\kappa) \\ (\vec{w}, \vec{\mu}, v, \rho) \in \boldsymbol{S} \iff \rho \in \boldsymbol{J}(\hat{N} \cdot 2^\kappa) \ \wedge\ \forall j\ w_j \in \boldsymbol{J}(2^\kappa) \end{cases}$$

We view Ξ_i as an *embedded* Schnorr protocol, cf. Definition 2.10, where the homomorphism (ϕ_1, ϕ_2) is augmented to (ϕ_1, ϕ_2, ϕ_3). Below, we show that non-batch protocol Ξ_i is secure under properties enforced by the protocol and suitable cryptographic assumptions. The security of the m-batch version of Ξ_i is a corollary of our main theorem in Section 5 (Theorem 5.1). To avoid significant overlap with the later sections, the proof of soundness below makes reference to the strong RSA assumption (Definition 4.7, Section 4.2) and the proof of Theorem 5.7, Section 5.2.1, and it is thus somewhat schematic. For full details, the reader is directed to the relevant references.

Claim 3.6. *Using the notation above, for some $\delta \in \mathsf{negl}$, the following holds true under strong RSA.*

1. *If $s_1, \ldots, s_\lambda \in \langle t \rangle$ then Ξ_i is γ-HVZK for $\gamma = (\lambda + 1) \cdot 2^{-\nu + 2\kappa} + 2^{-\kappa}$*
2. *If $N = p_1 p_2$ is Paillier-Blum s.t. $p_1, p_2 > q$, then, for $\pi_i \leftarrow \mathsf{ped}(1^\kappa)$, it holds that Ξ_i is δ-sound.*

Proof.
(**HVZK**) Since Ξ_i is an embedded protocol, we first define an efficient sampler $\mathcal{I}$ for the extended input $S = \phi_3(\vec{w}, \rho)$. First observe that S is $2^{-(\kappa+1)}$-close to a uniform element in the group $\langle t \rangle$ because $\rho \leftarrow \boldsymbol{I}(\hat{N} \cdot 2^\kappa)$ and $|\boldsymbol{I}(\hat{N} \cdot 2^\kappa)| > 2^{\kappa+1} \cdot |\langle t \rangle|$, where $|\langle t \rangle|$ is the size of the group. Consequently, the HVZK simulator can sample S by choosing an (almost) uniform element in $\langle t \rangle$; let $\mathcal{I}$ denote this process. Next we argue about the simulated transcript.

For $S \leftarrow \mathcal{I}$ as above, letting $(\vec{A}_1, \ldots, \vec{A}_\lambda, C) = (\phi_1(\vec{w}, \vec{\mu}), \phi_2(\vec{w}, v))$ denote the common input, recall that the simulator outputs $(\vec{B}_1, \ldots, \vec{B}_\lambda, D, T, e, \vec{z}, \vec{\beta}, v_0, \rho_0)$ where $(\vec{z}, \vec{\beta}, v_0, \rho_0) \leftarrow \boldsymbol{S}$, $e \leftarrow \boldsymbol{E}$ and $T = S^{-e} t^{\rho_0} \prod_{j=1}^\lambda s^{z_j}$

[17] Φ_π is a straightforward combination of ZK protocols found in [11] and [15].

"Online" Party $\mathcal{P}_\infty$		"Offline" Party $\mathcal{P}_0$
For $i \in [n]$, do:		
Sample $x_i, \alpha_i, u_i \leftarrow \mathbb{F}_q, Y_i \leftarrow \mathbb{G}$		Sample $x_0, \alpha_0, u_0 \leftarrow \mathbb{F}_q$, $Y_0 \leftarrow \mathbb{G}$
$\pi_i \leftarrow \mathsf{ped}(1^\kappa)$, $(N_i; p_i, q_i) \leftarrow \mathsf{gen}(1^\kappa)$.		$\pi_0 \leftarrow \mathsf{ped}(1^\kappa)$, $(N_0; p_0, q_0) \leftarrow \mathsf{gen}(1^\kappa)$,
Set $V_i = \mathcal{H}(\mathsf{aux}, \mathcal{P}_i, X_i, A_i, N_i, Y_i, \pi_i, u_i)$		Set $V_0 = \mathcal{H}(\mathsf{aux}, \mathcal{P}_0, X_0, A_0, N_0, W_0, Y_0, \pi_0, u_0)$
$(X_i, A_i) = (g^{x_i}, g^{\alpha_i})$		for $(X_0, A_0) = (g^{x_0}, g^{\alpha_0})$, $W_0 = \mathsf{enc}_0(x_0)$
	$\xrightarrow{V_1, \ldots, V_n}$ / $\xleftarrow{V_0}$	
	$\xrightarrow{(X_i, A_i, Y_i, N_i, \pi_i, u_i)_i}$ / $\xleftarrow{X_0, A_0, Y_0, N_0, W_0, \pi_0, u_0}$	
Verify V_0 and set $u = [\sum_{j=0}^n u_j]_q$,		$\forall i$ Verify V_i and set $u = [\sum_{j=0}^n u_j]_q$,
$Y = \prod_{j=0}^n Y_j$ and reassign $\mathsf{aux} := (\mathsf{aux}, u)$		$Y = \prod_{j=0}^n Y_j$ and reassign aux
$\forall i$, $\psi_i \leftarrow \Phi^{\mathsf{FS}}_{\pi_0}(N_i; p_i, q_i)$		$\forall i$, $\psi'_i \leftarrow \Phi^{\mathsf{FS}}_{\pi_i}(N_0, W_0, X_0; p_0, q_0, x_0)$
$e_i = \mathcal{H}(\mathcal{P}_i, X_i, A_i, u)$ and $z_i = [\alpha_i + e_i x_i]_q$		$e_0 = \mathcal{H}(\mathcal{P}_0, X_0, A_0, u)$ and $z_0 = [\alpha_0 + e_0 x_0]_q$
	$\xrightarrow{(\psi_i, z_i)_i}$ / $\xleftarrow{z_0, (\psi'_i)_i}$	
Calculate e_0 and check $g^{z_0} = A_0 \cdot X_0^{e_0}$		$\forall i$ Calculate e_i and check $g^{z_i} = A_i \cdot X_i^{e_i}$
Output $(Y, X_0, N_0, W_0, (X_i, N_i, \pi_i)_i; (x_i)_i)$		Output $(Y, X_0, N_0, W_0, \pi_0, X_\infty, (N_i, \pi_i)_i; x_0)$
		for $X_\infty = \prod_{i=1}^n X_i$

Figure 8: Threshold ECDSA, Key Generation (Σ_{kgen}) – The above protocol is essentially identical to the key-generation from [10]. In the first round, the parties generate their ECDSA key-shares X_i, their El-Gamal key-shares Y_i, their Paillier keys N_i and their Pedersen parameters; the offline party $\mathcal{P}_0$ also generates a Paillier ciphertext W_0 encrypting its *secret* ECDSA key-share $x_0 \in \mathbb{F}_q$. After exchanging decommitments in the second round, the parties generate proofs ψ_i, ψ'_i validating that their parameters are well-formed according to Definition 3.4; notice that $\mathcal{P}_0$'s proofs depend on identifier i because each ψ'_i depends on the i-th Pedersen tuple. Finally, in conjunction with the above, the parties execute an interactive variant of the Schnorr protocol for discrete logarithm, i.e. each tuple $(A_i, e_i = \mathcal{H}(\ldots), z_i)$ is a proof of knowledge for the discrete logarithm of X_i in base g and the proof is generated interactively over the three rounds.

"Online" Party $\mathcal{P}_\infty$		"Offline" Party $\mathcal{P}_0$
For $i \in [n]$, $j \in [\lambda]$, $\ell \in [m]$, do:		For $j \in [\lambda]$, $\ell \in [m]$, do:
$k_{i,j,\ell} \leftarrow \mathbb{F}_q,\ \vec{B}_{i,j,\ell} = \mathsf{com}_Y(k_{i,j,\ell})$		$\alpha_{j,\ell} \leftarrow \mathbb{F}_q,\ H_{j,\ell} = g^{\alpha_{j,\ell}^{-1}}$
$K_{i,\ell} = \mathsf{enc}_i(\sum_{j=1}^{\lambda} 2^{\tau(j-1)} \cdot k_{i,j,\ell})$		$C_\ell = \mathsf{enc}_0(\sum_{j=1}^{\lambda} 2^{\tau(j-1)} \cdot [\alpha_{j,\ell}^{-1}]_q)$
$\forall i,\ \psi_i \leftarrow \Xi_0^{\mathsf{FS}}((K_{i,j}, \vec{B}_{i,j,\ell})_{j,\ell}; (k_{i,j,\ell})_{j,\ell})$		$\forall i, \psi'_i \leftarrow \Xi_i^{\mathsf{FS}}((C_\ell, (H_{j,\ell})_j)_\ell; (\alpha_{j,\ell}^{-1})_{j,\ell})$
	$\xrightarrow{(\psi_i, (K_{i,\ell}, (\vec{B}_{i,j,\ell})_j)_\ell)_i}$	
		When obtaining $(\dots)_{i\in[n]}$
	$\xleftarrow{(\psi'_i)_i, (C_\ell, (H_{j,\ell})_j)_\ell}$	
$\forall i$ output		$\forall j, \ell$ set $\vec{B}_{j,\ell} = \prod_{i=1}^{n} \vec{B}_{i,j,\ell}$
$(k_{i,j,\ell}, \vec{B}_{1,j,\ell}, \dots, \vec{B}_{n,j,\ell}, H_{j,\ell})_{j,\ell}$		output $(\alpha_{j,\ell}, H_{j,\ell}, \vec{B}_{j,\ell})_{j,\ell}$

Figure 9: Threshold ECDSA, Presigning (Σ_{pres}) – In the first step, for identifier i and element ℓ in the batch, $\mathcal{P}_0$ obtains $(K_{i,\ell}, \vec{B}_{i,1,\ell}, \dots, \vec{B}_{i,m,\ell})$ where $\vec{B}_{i,j,\ell}$ is an El-Gamal commitment to some $k_{i,j,\ell} \in \mathbb{F}_q$ under public key $Y \in \mathbb{G}$ and $K_{i,\ell}$ is the packed ciphertext encrypting $\sum_{j=1}^{\lambda} 2^{\tau(j-1)} \cdot k_{i,j,\ell}$ under N_i. All tuples in the batch are accompanied by a single batch-proof ψ_i which validates that the tuples are well formed according to Ξ_0 as described in Definition 3.5; notice that Ξ_0 is an embedded Schnorr protocol augmented using the Pedersen parameters of $\mathcal{P}_0$. When obtaining the above, $\mathcal{P}_0$ calculates its contribution $H_{j,\ell}$ to the future nonce for each j and ℓ, as well as a packed ciphertext C_ℓ encrypting $\sum_{j=1}^{\lambda} 2^{\tau(j-1)} \cdot \alpha_{j,\ell}^{-1}$. Finally, viewing $H_{j,\ell}$ as the El-Gamal commitment $(\mathbb{1}, H_{j,\ell})$, $\mathcal{P}_0$ generates a batch-proof ψ'_i (one for each identifier i) validating that the tuples $(C_\ell, (H_{j,\ell})_j)_\ell$ are well formed according to Ξ_i as defined in Definition 3.5 and each ψ'_i depends on the Pedersen parameters of the i-th identifier.

mod $\hat{N}$, $D = C^{-e} \prod_{j=1}^{\lambda} (1 + 2^{\tau\cdot(j-1)} \cdot N)^{z_j} \cdot v_0^N \mod N^2$ and $\vec{B}_j = (\mathbb{1}, g)^{z_j} (g, Y)^{\beta_j} \cdot \vec{A}_j^{-e}$, for $j \in [\lambda]$. Notice that the simulated $\vec{\beta}$ and v_0 are identically distributed with the real values and that each $z_1, \dots, z_\lambda$ and ρ_0 is $(2^{-\nu+2\kappa})$-close to the corresponding real value because $|\boldsymbol{E}| \cdot |\boldsymbol{I}|/|\boldsymbol{J}| = 2^{-\nu+2\kappa}$. Thus the claimed HVZK follows by triangle inequality.

(**Soundness**) Define $\boldsymbol{V} = \{(e, e') \in \boldsymbol{I}(2^\kappa)^2 \text{ s.t. } (e-e') \notin \{-1, 0, 1\} \text{ and } p_1, p_2 \nmid (e-e')\}$. Under the strong RSA assumption, using the same analysis as Theorem 5.7, Section 5.2.1, letting $\tau = (\dots, e, \vec{z}, \vec{\beta}, v_0, \rho_0)$ and $\tau' = (\dots, e', \vec{z}', \vec{\beta}', v'_0, \rho'_0)$ denote two suitable transcripts s.t. $(e, e') \in \boldsymbol{V}$, we can extract all the witnesses $\vec{w}, \vec{\mu}, v, \rho$ with probability $1-\varepsilon$ over the choice of $\pi_i \leftarrow \mathsf{ped}(1^\kappa)$, for $\varepsilon \in \mathsf{negl}$. In particular, $\vec{w} = (\vec{z}-\vec{z}')/(e-e')$ and $\rho = (\rho_0 - \rho'_0)/(e - e')$ over $\mathbb{Z}$ (the integers). Furthermore, since $\vec{z}, \vec{z}' \in \boldsymbol{J}(2^\kappa)^\lambda$, $\rho_0, \rho'_0 \in \boldsymbol{J}(\hat{N} \cdot 2^\kappa)$ and $|e - e'| \geq 2$, we have $\vec{w} \in \boldsymbol{J}(2^\kappa)^\lambda$ and $\rho \in \boldsymbol{J}(\hat{N} \cdot 2^\kappa)$, i.e. $\vec{w}$ and ρ are in the right range. To conclude, notice that $|\boldsymbol{V}|/|\boldsymbol{E}|^2 \leq 1/2^{\kappa-3} \in \mathsf{negl}$ since N does not admit small factors (smaller than $q \approx 2^\kappa$). □

3.3 Signing

Without loss of generality, we assume that the signing phase consumes all the available presignatures. Further, recall that the parties start with the same message to be signed, i.e. we are agnostic about how the parties reach consensus on messages $\{\mathrm{msg}_{j,t}\}_{j,t}$ and we write $m_{j,t} = \mathcal{F}(\mathrm{msg}_{j,t})$, where $\mathcal{F}$ is the internal hash function of ECDSA. The Schnorr protocols from Σ_{sign} are described in Definitions 3.7, 3.8 and 3.10. We note that Θ_R and Θ'' have perfect HVZK and soundness $1/q$ (we do not prove this fact).

3.3.1 `init-tecdsa` functionality

The signing process is the only component of the protocol where the MPC among the cosignatories is not trivial. Specifically the parties execute an interactive protocol for calculating the `init-tecdsa` functionality described below (c.f. Figure 10). By setting $H = g$, we recall that `init-tecdsa` is a distributed variant of the ECDSA functionality for public key $X_\infty = \prod_{i=1}^{n} X_i$. As such, by allowing the parties to change the base-point from g to H, any threshold-ECDSA protocol from the literature can be tweaked to realize `init-tecdsa`. Herein, we implement the functionality via [11], aka the CMP protocol, and we give full details in Figure 15, Appendix A .

FIGURE 10 (`init-tecdsa` functionality)

Common Input. $(\vec{B}_i, X_i)_{i\in[n]} \in \mathbb{G}^{3n}$ and $H \in \mathbb{G}$.

Secret Input. Each $\mathcal{P}_i$ holds input (k_i, ρ_i, x_i) s.t. $\vec{B}_i = (g, Y)^{\rho_i} \cdot (\mathbb{1}, g)^{k_i}$ and $X_i = g^{x_i}$.

Operation. If the secret input is consistent with the public input do:

(a) Set $k = \sum_{i=1}^{n} k_i \mod q$ and set $R = H^{k^{-1}} \in \mathbb{G}$.

(b) Sample random $\{\chi_i \in \mathbb{F}_q\}_{i=1}^{n}$ subject to $\sum_{i=1}^{n} \chi_i = k \cdot \sum_{i=1}^{n} x_i \mod q$.

Output: (R, χ_i, r) to each $\mathcal{P}_i$, where $r = R|_{\text{x-axis}}$.

Figure 10: `init-tecdsa` functionality

3.3.2 ZK for Signing

Definition 3.7. Define Θ_R to be the Schnorr protocol associated with $(\phi, \boldsymbol{E})$ for $\boldsymbol{E} = \boldsymbol{I}(2^\kappa)$ and

$$\begin{aligned}\phi : \mathbb{F}_q^2 &\to \mathbb{G}^3\\ (k, b) &\mapsto (g^b, Y^b \cdot g^k, R^b)\end{aligned}$$

Definition 3.8. For $(\hat{N}, t, s_1, r_1, \ldots) = \pi_0$, define Schnorr protocol Θ' for $(\phi, \boldsymbol{E}, \boldsymbol{R}, \boldsymbol{S})$ where $\boldsymbol{E} = \boldsymbol{I}(2^\kappa)$

$$\begin{aligned}\phi : \mathbb{Z}^{2\lambda} \times \mathbb{F}_q^{2\lambda} \times \mathbb{Z}_{N_0}^* \times \mathbb{Z} &\to \mathbb{G}^{4\lambda} \times \mathbb{Z}_{N_0^2}^* \times \mathbb{Z}_{\hat{N}}^*\\ (\vec{w}, \vec{\mu}, \vec{z}, \vec{\gamma}, v, \rho) &\mapsto (\phi_1(\vec{w}, \vec{\mu}), \phi_1(\vec{z}, \vec{\gamma}), \phi_2(\vec{w}, \vec{z}, v), \phi_3(\vec{w}, \vec{z}, \rho))\end{aligned}$$

and

$$\begin{cases}\phi_1 : (\vec{w}, \vec{\mu}) \mapsto (g^{\mu_j}, Y^{\mu_j} \cdot g^{w_j})_{j=1}^{\lambda}\\ \phi_2 : (\vec{w}, \vec{z}, v) \mapsto \prod_{j=1}^{\lambda} (1 + 2^{\tau\cdot(j-1)} \cdot N_0)^{w_j} \cdot W_0^{z_j} \cdot v^{N_0} \mod N_0^2\\ \phi_3 : (\vec{w}, \rho) \mapsto t^\rho \cdot \prod_{j=1}^{\lambda} s_j^{w_j} r_j^{z_j} \mod \hat{N}\end{cases}$$

with

$$\begin{cases}(\vec{w}, \vec{\mu}, \vec{z}, \vec{\gamma}, v, \rho) \in \boldsymbol{R} \iff \rho \in \boldsymbol{I}(\hat{N} \cdot 2^\kappa) \ \wedge \ \forall j \ (z_j \in \boldsymbol{I}(2^\kappa) \wedge w_j \in \boldsymbol{I}(2^{\kappa+\nu}))\\ (\vec{w}, \vec{\mu}, \vec{z}, \vec{\gamma}, v, \rho) \in \boldsymbol{S} \iff \rho \in \boldsymbol{J}(\hat{N} \cdot 2^\kappa) \ \wedge \ \forall j \ (z_j \in \boldsymbol{J}(2^\kappa) \ \wedge \ w_j \in \boldsymbol{J}(2^{\kappa+\nu}))\end{cases}$$

Similarly to Ξ_i from Definition 3.5, we view Θ' as an *embedded* Schnorr protocol, cf. Definition 2.10, where the homomorphism (ϕ_1, ϕ_1, ϕ_2) is augmented to $(\phi_1, \phi_1, \phi_2, \phi_3)$, and the security of the batch protocol is a corollary of Theorem 5.1 . Below, in Claim 3.9, we state the security properties of (non-batch) Θ'; we do not provide a proof for the claim since it is essentially identical to the proof of Claim 3.6.

Claim 3.9. *Using the notation above, for some $\delta \in \mathsf{negl}$, the following holds true under strong RSA.*

1. *If $s_1, \ldots, s_\lambda \in \langle t \rangle$ then Ξ_i is γ-HVZK for $\gamma = (2\lambda + 1) \cdot 2^{-\nu+2\kappa} + 2^{-\kappa}$*
2. *If $N = p_1p_2$ is Paillier-Blum s.t. $p_1, p_2 > q$, then, for $\pi_0 \leftarrow \mathsf{ped}(1^\kappa)$, it holds that Θ' is δ-sound.*

Definition 3.10. Define Θ'' to be the Schnorr protocol associated with tuple $(\phi, \boldsymbol{E})$ for $\boldsymbol{E} = \mathbb{F}_q$, $\phi : \mathbb{F}_q^3 \to \mathbb{G}^4$ s.t. $(k, b, v) \mapsto (g^b, Y^b \cdot g^k, g^v, Y^v \cdot X_\infty^k)$.

Remark 3.11 (Communication Complexity). Ignoring very small constant overheads, each $(S_\ell)_\ell$, ψ' and $(U_{j,\ell}, r_{j,\ell}, \psi_{j,\ell})_{j,\ell}$ has bit-length $m\log(N^2)$, $m\log(\hat{N})$ and $\lambda m 6\log(q)$ respectively. Thus, amortized over the number of signatures, $\mathcal{P}_0$ recieves a message of bit-length $\mathsf{cc} = 1/\lambda \cdot (\log(\hat{N}) + 2\log(N)) + 6\log(q)$. So, for $q \approx 2^{256}$, using $\hat{N}, N \approx 2^{2048}$ and $\lambda = 3$, we get $\mathsf{cc} \approx 320$B. Using $\hat{N}, N \approx 2^{4096}$, and $\lambda = 6$, we get $\mathsf{cc} \approx 288$B.

“Online” Party $\mathcal{P}_\infty$ — “Offline” Party $\mathcal{P}_0$

Retrieve $(\vec{B}_{i,j,\ell})_{i,j,\ell}$ and $(H_{j,\ell}, k_{i,j,\ell})_{j,\ell}$ and do:

Initialize `tecdsa` *Functionality : (Figure 10 & Figure 15)*

$\forall j, \ell$ run `init` on $((\vec{B}_{i,j,\ell}, X_i)_i, H_{j,\ell}); (k_{i,j,\ell}, x_i)_i)$

Obtain $R_{j,\ell}, r_{j,\ell}$ and $(\chi_{i,j,\ell})_i$ and set $(\vec{U}_{i,j,\ell} = \mathsf{com}_Y(\chi_{i,j,\ell}))_i$

Prepare Payload:

$\forall i$ set $\zeta_{i,j,\ell} = [k_{i,j,\ell} \cdot r_{j,\ell}]_q$ and $\eta_{i,j,\ell} \leftarrow \boldsymbol{I}(2^\varepsilon)$

$$\mu_{i,j,\ell} = [r_{j,\ell} \cdot \chi_{i,j,\ell} + m_{j,\ell} \cdot k_{i,j,\ell}]_q + q \cdot \eta_{i,j,\ell}$$

$$S_{i,\ell} = [\textstyle\prod_{j=1}^{\lambda} W_0^{2^{\tau(j-1)}\zeta_{i,j,\ell}} \cdot \mathsf{enc}_0(2^{\tau(j-1)} \cdot \mu_{i,j,\ell})]_{N_0^2}$$

Aggregate-Prove: (Figure 7)

$\psi_{j,\ell} \leftarrow \Theta^{\mathtt{AGT}}_{R_{j,\ell}}((\vec{B}_{j,\ell}, H_{j,\ell}); (k_{i,j,\ell}))_i$ for $\vec{B}_{j,\ell} = \prod_i \vec{B}_{i,j,\ell}$

$\psi' \leftarrow \Theta'^{\mathtt{AGT}}(((\vec{V}_{j,\ell}, \vec{Z}_{j,\ell})_j, S_\ell)_\ell; (\zeta_{i,j,\ell}, \mu_{i,j,\ell})_{j,\ell})$

$\psi'' \leftarrow \Theta''^{\mathtt{AGT}}((\vec{B}_{j,\ell}, \vec{U}_{j,\ell})_{j,\ell}; (k_{i,j,\ell})_{i,j,\ell})$

for $\vec{V}_{j,\ell} = \prod_i \vec{B}_{i,j,\ell}^{r_{j,\ell}}$ and $S_\ell = \prod_i S_{i,\ell}$

$\vec{U}_{j,\ell} = \prod_i \vec{U}_{i,j,\ell}$ and $\vec{Z}_{j,\ell} = \prod_i \vec{B}_{i,j,\ell}^{m} \cdot \vec{U}_{i,j,\ell}^{r_{j,\ell}}$

$\xrightarrow{\psi', \psi'', ((r_{j,\ell}, U_{j,\ell}, \psi_{j,\ell})_j, S_\ell)_\ell}$

$\forall j, \ell$ decode $\hat{\sigma}_{j,\ell}$ s.t.

$$\mathsf{dec}_0(S_j) = \textstyle\sum_{j=1}^{\lambda} 2^{\tau(j-1)} \hat{\sigma}_{j,\ell}$$

Set $\sigma_{j,\ell} = [\alpha_{j,\ell} \cdot \hat{\sigma}_{j,\ell}]_q$

$\forall \ell, j$ output $(r_{j,\ell}, \sigma_{j,\ell})$

Figure 11: Threshold ECDSA, Signing (Σ_{sign}) – From the offline party's perspective, $\mathcal{P}_0$ obtains nonces $R_{j,\ell}$ and commitments $\vec{U}_{j,\ell}$ to $\chi_{j,\ell}$ and ciphertexts S_ℓ. These values are accompanied by proofs $\psi_{j,\ell}$ and ψ', ψ'' (c.f. Definition 3.7, 3.8 and 3.10) which validate the following: $\psi_{j,\ell}$ validates that $r_{j,\ell}$ is well-formed against $H_{j,\ell}$ and $\vec{B}_{j,\ell}$. Batch-proof ψ'' validates that each $\vec{U}_{j,\ell}$ is an El-Gammal commitment to $\chi_{j,\ell} = x_\infty \cdot k_{j,\ell}$ for all j, ℓ. Batch-proof ψ' validates that each S_ℓ is a packed ciphertext encrypting $\sum_j 2^{\tau(j-1)}\sigma_{j,\ell}/\alpha_{j,\ell}$ where $\sigma_{j,\ell}$ is a valid signature for $m_{j,\ell}$ and nonce $r_{j,\ell}$, for all ℓ. For the online party, the protocol consists of the following three-step process. First, $\mathcal{P}_\infty$ calculates the “initialization” data $r_{j,\ell}$ and $\vec{U}_{j,\ell}$ as per the V-MPC functionality in Figure 10, for every j and ℓ. Then, for each $\ell \in [m]$, $\mathcal{P}_\infty$ calculates the “payload” $S_{i,\ell}$ encrypting the packed partial signatures for each identifier i, and, lastly, $\mathcal{P}_\infty$ generates the accompanying proofs and sends the data to $\mathcal{P}_0$. We note that the proofs $\{\psi_{j,\ell}\}_{j,\ell}$ are not batchable because each proof corresponds to a different homomorphism depending on $R_{j,\ell}$. When virtualizing $\mathcal{P}_\infty$, for the `init` phase, it suffices to run *any* threshold-ECDSA protocol (with appropriate tweaks) because, as mentioned multiple times, almost any threshold-ECDSA procol can be tweaked to realize the `init-tecdsa` functionality. For the proofs, the parties are instructed to run the proof-aggregation protocol from (Figure 7) .

4 Security

Theorem 4.1. *Under suitable cryptographic assumptions, it holds that Σ_{ecdsa} UC-realizes functionality $\mathcal{F}_{\mathsf{tsig}}$ in the presence of a global random oracle functionality $\mathcal{H}$.*

Our main theorem is a corollary of Theorems 4.5 and 4.8.

4.1 Unforgeability & Simulatability imply UC Security

Let SGN denote a signature scheme and let Σ be a threshold protocol for SGN. We show that if Σ and SGN satisfy some limited security requirements, then Σ UC-realizes $\mathcal{F}_{\text{tsig}}$ in the strict global random oracle model. We begin by defining the aforementioned security requirements.

Let $\mathcal{A}$ denote an adaptive adversary and write $\text{Real}^{\mathcal{H}}_{\Sigma,\mathcal{A}}(1^\kappa, z)$ for the adversary's view in an execution of Σ in the presence of an *adaptive* PPTM adversary $\mathcal{A}$ given auxiliary input z. Recall that $\mathcal{H}$ denotes the random oracle. Without loss of generality assume that $\text{Real}^{\mathcal{H}}_{\Sigma,\mathcal{A}}(1^\kappa, z) = (\mathsf{pk}^{\Sigma}, \ldots)$, where pk^{Σ} denotes the public key resulting from the execution of Σ. It is assumed that $\mathcal{A}$ chooses the messages for signing in Σ. Next, for an oracle-aided algorithm $\mathcal{S}$ with black-box access to $\mathcal{A}$ and oracle access to $\mathcal{G}$ and $\mathcal{H}$, write $\text{Ideal}^{\mathcal{H}}_{\mathcal{G},\mathcal{S}}(1^\kappa, z) = (\mathsf{pk}^{\mathcal{G}}, \text{Out}^{\mathcal{S}})$ for the pair of random variable consisting of the public key generated by $\mathcal{G}$ and the simulator's output.

Remark 4.2. In the above, the adversary chooses the messages for signing analogously to the environment accessing the input-output tape of the parties in the UC experiment. Furthermore, it is assumed that the parties agree on the message to be signed; this assumption does not incur any loss of generality since this can be enforced via a consensus mechanism.

Definition 4.3 ($\mathcal{G}$-Simulatability)**.** Using the notation above, we say that Σ is $\mathcal{G}$-simulatable in the ROM if the following holds for every adversary $\mathcal{A}$ and every $\varepsilon \in 1/\mathsf{poly}$. There exists a simulator $\mathcal{S}$ with oracle access to $\mathcal{G}$ and $\mathcal{H}$, and black-box access to $\mathcal{A}$, such that:

1. $\mathcal{G}$ is queried by $\mathcal{S}$ only on messages intended for signing as prescribed by Σ, and chosen by $\mathcal{A}$.

2. If $\mathcal{A}$ does not corrupt all parties in some $\boldsymbol{Q} \in \mathsf{QRMs}$ simultaneously in any given epoch, then

$$\{\text{Real}^{\mathcal{H}}_{\Sigma,\mathcal{A}}(1^\kappa, z)\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*} \stackrel{\varepsilon}{\equiv} \{\text{Ideal}^{\mathcal{H}}_{\mathcal{G},\mathcal{S}}(1^\kappa, z)\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*} \tag{4}$$

Remark 4.4 (Simulatability vs Standalone Security). We observe that the notion of simulatability is quite close to the MPC security notion of "standalone security". We recall that a protocol Π standalone-realizes $\mathcal{F}$ if for every adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that for all $\varepsilon \in 1/\mathsf{poly}$ the joint distribution of the view of $\mathcal{A}$ together with the honest parties' outputs in an execution of the protocol is $1/\varepsilon$-indistinguishable from the output of $\mathcal{S}$ together with the honest parties' outputs when interacting with a trusted party computing $\mathcal{F}$. So, viewing the signing oracle $\mathcal{G}$ as the ideal functionality, we note that simulatability is a weaker notion than standalone security because the simulator may depend on the distinguishing advantage; standalone security stipulates that there exists a single PPTM $\mathcal{S}$ for every $\varepsilon \in 1/\mathsf{poly}$, i.e. the order of the quantifiers is reversed.

Theorem 4.5. *Let* SGN *denote a signature scheme and let* Σ *denote a threshold-*SGN *protocol. Let* $\mathcal{G}$ *denote an augmented signature oracle (cf. Figure 3) such that*

1. SGN *is* $\mathcal{G}$*-existentially unforgeable according to Definition 2.2.*

2. Σ *is* $\mathcal{G}$*-simulatable in the ROM according to Definition 4.3.*

Then, Σ *UC-realizes* $\mathcal{F}_{\text{tsig}}$ *in the strict global random oracle model.*

Proof. (**UC Simulation**) First, we describe the UC simulation, which is trivial. The simulator simply runs the code of the honest parties as prescribed by Σ, i.e. the simulator samples all the secrets as prescribed and plays against the adversary in exactly the same way as in the real execution. To interact with the functionality, the simulator proceeds as follows:

1. After key generation, the simulator submits the verification algorithm $\mathcal{V}$ which depends on the public key pk calculated by $\mathcal{S}$ during the simulation Σ_{kgen}. If $\mathcal{S}$ fails to reconstruct pk, e.g. because $\mathcal{Z}$ aborted certain parties or one of the decommitments failed, then $\mathcal{S}$ is instructed to halt.

2. Every time a signature is calculated during the simulation $\Sigma_{\mathtt{sign}}$, then $\mathcal{S}$ submits the resulting signature-string to the functionality. (The simulator does not interact with the functionality if it fails to calculate the signature, or during the simulation of Σ_{pres})

3. Depending on $\mathcal{Z}$'s corruption pattern and the protocol's key-refresh schedule, the simulator registers parties as corrupted and/or quarantined. In other words, if $\mathcal{Z}$ decides to corrupt or decorrupt a party $\mathcal{P}_i$, then $\mathcal{S}$ informs the functionality via the relevant interface, and, after the simulation of Σ_{refr}, all decorrupted parties are marked as honest.

It is not hard to see that the above simulation is *perfect* unless $\mathcal{Z}$ can forge signatures in the protocol (i.e. in the real world). Formally, using the notation from Figure 14, there is a PPTM $\mathcal{X}$ (not necessarily one of the $\mathcal{P}_i$'s) that verifies the tuple $(sid, m, \sigma, \mathsf{pk})$ for a message m which was never signed before. In the ideal world, $\mathcal{X}$ queries $\mathcal{F}_{\mathsf{tsig}}$ which returns `false` to $\mathcal{X}$ since m was never signed before. In the real world, however, $\mathcal{X}$ runs the verification algorithm $\mathcal{V}$ associated with pk and outputs `true` if σ is a valid forgery. In turn, this allows $\mathcal{Z}$ to distinguish the real and ideal experiment. Thus, by viewing $\mathcal{Z}$ and $\mathcal{X}$ as a single PPTM, there exists a PPTM $\mathcal{A}_0$ corrupting a subset of parties in Σ that outputs a forgery in an execution of Σ with noticeable probability, say $\alpha \notin \mathsf{negl}$. We will show that since Σ is simulatable we can use $\mathcal{A}_0$ to break the unforgeability assumption on SGN (Item 1 in Theorem 4.5).

(**Reduction to Unforgeability.**) By definition, there exists $\beta \in 1/\mathsf{poly}$ such that $\beta(\kappa) \leq \alpha(\kappa)$ for infinitely many κ's. Take $\varepsilon = \beta/2$ and fix $\mathcal{S}$ as per Definition 4.3. Deduce that $\mathrm{Ideal}^{\mathcal{H}}_{\mathcal{G},\mathcal{S}}(1^\kappa, z)$ contains a forgery with probability at least $\alpha/2 \notin \mathsf{negl}$ since, by Equation (4), $\mathrm{Ideal}^{\mathcal{H}}_{\mathcal{G},\mathcal{S}}(1^\kappa, z)$ contains a forgery with probability is at least $\alpha - \varepsilon \geq \alpha - \beta/2 \geq \alpha/2$. In turn, this implies that $\mathcal{G}$ is useful for forging SGN signatures, in contradiction with the hypothesis of the theorem. □

On the Random Oracles & Rewinding. Recall that the oracle is *strict* in the UC definition (Figure 5), i.e. $\mathcal{S}$ cannot tamper with the adversary's queries; it cannot even observe these queries. Using the jargon from the literature, the oracle in the UC-ideal experiment is *non-programmable* and *non-observable*. In contrast, in the definition of simulatability, the adversary queries the random oracle *through* the simulator (i.e. $\mathcal{S}$ also simulates the oracle in the experiment). It may appear strange that we handle the oracle differently in each case, so we offer the following remark that clarifies this discrepancy (we also touch on rewinding in the remark).

Remark 4.6. Recall that our UC simulator is essentially trivial and it does not interfere with the random oracle and it is straightline (non-rewinding). Next, in Theorem 4.5, specifically in the reduction to unforgeability, note that the simulator has read/write-access to $\mathcal{A}_0$'s oracle tape (this is a truism of run-of-the-mill reductions). As a result, if our trivial UC simulator fails for some $\mathcal{Z}$, then the following PPTM $\mathcal{B}$ breaks the unforgeability of SGN: $\mathcal{B}$ simply runs the simulator from Definition 4.3 on $\mathcal{A}_0$, where $\mathcal{A}_0$ is the well-defined PPTM mentioned in the above proof. In particular, (i) $\mathcal{B}$ carefully tinkers with the oracle (as prescribed by Definition 4.3), and (2) $\mathcal{B}$ may also *rewind* $\mathcal{A}_0$ at will (another truism of conventional reductions).

4.2 Simulatability of Σ_{ecdsa}

In Figure 12, we define the *enhanced* signing oracle for ECDSA which gives rise to the notion of *enhanced unforgeability* according to Definition 2.2. We note that enhanced unforgeability has been studied by Canetti et al. [11] in the generic group model (GGM) and the ROM, where they rule out any efficient attack in this model, and, recently, by Groth and Shoup [27] who provide a more fine-grained analysis in the GGM. Next we define strong-RSA, desisional Diffie-Hellman (DDH), desisional composite residuocity (DCR).

Let DDH, sRSA and DCR denote the following distriburions. $(N, C) \leftarrow \mathsf{sRSA}(1^\kappa)$ where N is a safe biprime of size $O(\kappa)$ 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 $(\mathbb{G}, g, q)$ generated by 1^κ, and $(N, [(1+N)^z \cdot \rho^N]_{N^2}, z) \leftarrow \mathsf{DCR}(1^\kappa)$ for $z \leftarrow \{0,1\}$ and $\rho \leftarrow \mathbb{Z}_N^*$.

Definition 4.7. 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 $[m^e = C]_N$.
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 $z = \beta$

Theorem 4.8. *Assuming DDH, DCR, and strong RSA, it holds that Σ_{ecdsa} is $\mathcal{G}^*$-simulatable.*

FIGURE 12 (Enhanced Signing Oracle $\mathcal{G}^*$ for ECDSA)

Parameters. Hash function $\mathcal{F} : \{0,1\}^* \to: \{0,1\}^*$.

Operation.

1. On input $(\mathsf{gen}, (\mathbb{G}, g, q))$, sample $\mathsf{sk} = x \leftarrow \mathbb{F}_q$ and return $\mathsf{pk} = X = g^x$.

   Store $(\mathsf{sk}, \mathsf{pk})$ in memory and ignore future calls to gen.

2. On input pres, sample $k \leftarrow \mathbb{F}_q$ and return $R = g^{k^{-1}}$.

   Store $(R; k)$ in memory and standby.

3. On input $(\mathsf{sign}, \text{msg}, R)$, do:

   (a) Retrieve $(R; k)$ from memory.

   If no such R exists or R is undefined, sample $k \leftarrow \mathbb{F}_q$ and (re)assign $R := g^{k^{-1}}$.

   (b) Set $\sigma = [k(m + rx)]_q$ where $m = \mathcal{F}(\text{msg})$ and $r = R|_{\text{x-axis}}$.

   (c) Erase $(R; k)$ from memory and return (r, σ).

Figure 12: Enhanced Signing Oracle $\mathcal{G}^*$ for ECDSA

4.2.1 Proof of Theorem 4.8

We will show that for every $\varepsilon \in 1/\mathsf{poly}$, there exists a simulator $\mathcal{S}$ that interacts with $\mathcal{A}$ as per Definition 4.3. Our simulator is parameterized by $r \in \mathsf{poly}$ to be determined by the analysis (and r depends on ε). Next we give a high-level description of $\mathcal{S}$ and we refer the reader to in Figure 13 for the detailed description. At the beginning of simulation, $\mathcal{S}$ chooses a random non-corrupted party, dubbed the *special* party, and all other non-corrupted parties are simulated by running their code as prescribed. To deal with adaptive corruptions, if the adversary decides to corrupt the special party the the simulation is reset, via rewinding, and the special party is chosen afresh. Furthermore, we note that $\mathcal{S}$ simulates the random oracle and $\mathcal{A}$ queries $\mathcal{S}$ when it needs to query $\mathcal{H}$. In particular, in Figure 13, every time $\mathcal{S}$ "retrieves" a value, we mean that it obtains the relevant value from $\mathcal{A}$'s queries, and, the simulated message of $\mathcal{S}$ are consistent with the simulated oracle (by programming the simulated oracle accordingly). Then, at a high level, our simulator proceeds as follows:

1. Invoke $\mathcal{G}$ to obtain a public key X and run the protocol with $\mathcal{A}$ to land on key X by suitably choosing the special party's message, i.e. set $X_b = X \cdot (\prod_{j \neq b} X_j)^{-1}$.

2. Extract $\mathcal{A}$'s Paillier keys and ECDSA key shares by rewinding.

   If the adversary aborts during the first run, i.e. by quitting or returning an invalid proof, then the simulator halts. Else, $\mathcal{S}$ rewinds the adversary until it obtains a second valid transcript to extract the secrets and continue the rest of the simulation on the first valid execution, i.e. rewind one last time (if extraction fails after r attempts, halt).

3. To calculate the special party's message do:

   (a) Extract the adversary's randomness by decrypting the relevant Paillier messages (encrypted under keys chosen by the adversary).

   (b) Calculate the relevant message by (i) encrypting zeros under the Paillier and El-Gamal keys for hidden data, e.g. $\mathcal{P}_b$'s Paillier ciphertexts, and (ii) using the extracted randomness above and queries to $\mathcal{G}$ for non-hidden data, e.g. the output signature.

To show that the above simulation is indistinguishable from the real protocol, we define two experiments (hybrids) where the first experiment coincides with the simulated execution and the second experiment coincides with the real execution. Namely:

Experiment 1. The first experiment is identical to Figure 13 except that $\mathcal{S}$ emulates $\mathcal{G}$ locally.

Experiment 2. The second experiment is identical to the above except that $\mathcal{S}$ uses the right Paillier & El-Gamal ciphertexts, instead of zeroes; $\mathcal{S}$ can do this because it has access to all the secrets.

FIGURE 13 ($\mathcal{G}^*$-simulation for Σ_{ecdsa})

Parameters. Adversary $\mathcal{A}$ and RO $\mathcal{H}$.

Operation.

init. Call $\mathcal{G}^*$ on input $(\mathbb{G}, g, q)$. Obtain $\mathsf{pk} = X$.

– (Σ_{kgen}) Choose $\mathcal{P}_b \leftarrow \boldsymbol{H} = \boldsymbol{P} \setminus \boldsymbol{C}$ and do:

1. Hand over $V_b \leftarrow \{0,1\}^*$ to $\mathcal{A}$.
2. When obtaining $(V_j)_{j \neq b}$, retrieve $(X_j, A_j, Y_j, N_j, \pi_j, u_j)_{j \neq b}$ and do:
   (a) Set $X_b = X \cdot (\prod_{j \neq b} X_j)^{-1}$ and $Y_b = g^y \cdot (\prod_{j \neq b} Y_j)^{-1}$ for $y \leftarrow \mathbb{F}_q$.
   (b) Sample $z_b, e_b \leftarrow \mathbb{F}_q$ and set $A_b = X_b^{e_b} \cdot g^{-z_b}$. If $b = 0$, set $W_0 = \mathsf{enc}_0(0)$.
   Calculate all other values as prescribed.
   Hand over $(X_b, A_b, Y_b, N_b, W_b, \pi_b, u_b)$ to $\mathcal{A}$ (where $W_b = \emptyset$ if $b \neq 0$).
3. When obtaining $(X_j, A_j, \ldots)_{j \neq b}$, do:
   (a) Calculate ψ_b (or $\psi'_1, \ldots, \psi'_n$ if $b = 0$) by invoking the HVZK simulator.
   Hand over z_b, ψ_b (or $z_0, (\psi'_j)_{j \neq 0}$ if $b = 0$) to $\mathcal{A}$.
4. Rewind $\mathcal{A}$ by providing fresh u_b and e_b to extract (x_j, p_j, q_j) such that $(X_j, N_j) = (g^{x_j}, p_j q_j)$.

– (Σ_{pres}) Make $m \cdot \lambda$ calls to $\mathcal{G}^*$ on input pres. Obtain $(R_{j,\ell})_{j,\ell} \in \mathbb{G}^{m \cdot \lambda}$ and do:

1. If $\mathcal{P}_b \neq 0$, sample $\vec{B}_{b,j,\ell} \leftarrow \mathbb{G}^2$ and set $K_{b,j} = \mathsf{enc}_b(0)$ and calculate ψ_b using the HVZK simulator.
   Hand over $\psi_b, (K_{b,j}, (\vec{B}_{b,j,\ell})_\ell)_j$ to $\mathcal{A}$.
   When obtaining $(\psi_i)_{i \neq 0}, (C_j, (H_{j,\ell})_\ell)_j$ extract $\{\alpha_{j,\ell}\}_{j,\ell}$ by decrypting $\{C_j\}_j$.
2. Else, when obtaining $(\psi_i, (K_{i,j}, (\vec{B}_{i,j,\ell})_\ell)_j)_{i \neq b}$, extract $\{k_{i,j,\ell}\}_{i,j,\ell}$ by decrypting $\{K_{i,j}\}_{i,j}$ and do
   (a) Set $H_{j,\ell} = R_{j,\ell}^{k_{j,\ell}}$ for $k_{j,\ell} = \sum_{i \neq 0} k_{i,j,\ell}$.
   (b) Set $C_j = \mathsf{enc}_0(0)$ and calculate $(\psi'_i)_{i \neq 0}$ using the HVZK simulator.
   Hand over $(\psi_i)_{i \neq 0}, (C_j, (H_{j,\ell})_\ell)_j$ to $\mathcal{A}$.

– (Σ_{sign}) Make $m \cdot \lambda$ calls to $\mathcal{G}^*$ on input $(\mathsf{sign}, \mathrm{msg}_{j,\ell}, R_{j,\ell})$. Obtain $(r_{j,\ell}, \sigma_{j,\ell})_{j,\ell} \in \mathbb{F}_q^{2m \cdot \lambda}$ and do:

If $\mathcal{P}_b = \mathcal{P}_0$, when obtaining $\psi', \psi'', ((r_{j,\ell}, U_{j,\ell}, \psi_{j,\ell})_\ell, S_j)_j$ from $\mathcal{A}$, output $(r_{j,\ell}, \sigma_{j,\ell})_{j,\ell}$ for $\mathcal{P}_0$.

Else, if $\mathcal{P}_b \neq \mathcal{P}_0$, retrieve $\{\alpha_{j,\ell}\}_{j,\ell}$, set $m_{j,\ell} = \mathcal{H}(\mathrm{msg}_{j,\ell})$, and do:

1. Obtain $(\chi^*_{j,\ell} = \sum_{i \neq b} \chi_{i,j,\ell})_{j,\ell}$ from the Virtual Party MPC call (e.g. run sim for $\Sigma_{\mathtt{cmp}}$ in Figure 16).
   Set $\rho_{j,\ell} = [\chi^*_{j,\ell} r_{j,\ell} + \sum_{i \in [n] \setminus \{b\}} k_{i,j,\ell}(r_{j,\ell} x_0 + m_{j,\ell})]_q$.
2. Set $S_{b,j} = \mathsf{enc}_0(\sum_\ell 2^{\tau(\ell-1)}([\sigma_{j,\ell}/\alpha_{j,\ell} - \rho_{j,\ell}]_q + q \cdot \eta_{b,j,\ell}))$ for $\eta_{b,j,\ell} \leftarrow \boldsymbol{I}(2^\nu)$ and $\vec{U}_{j,\ell} \leftarrow \mathbb{G}^2$
3. Run the simulation for proof aggregation for Θ, Θ' and Θ'' (Claim 2.16).
   When obtaining $\{\vec{U}_{i,j,\ell} = (U_{i,j,\ell}, U'_{i,j,\ell})\}_{i \neq b}$, set $\vec{U}_{b,j,\ell} = \vec{U}_{j,\ell} \cdot (\mathbb{1}, g^{-\chi^*_{j,\ell}}) \prod_{i \neq b} (U_{i,j,\ell}, U^y_{i,j,\ell})^{-1}$.
   Simulate ψ'' according to $\vec{U}_{b,j,\ell}$

Figure 13: $\mathcal{G}^*$-simulation for Σ_{ecdsa}

Clearly, the first experiment is identical to the simulation. Furthermore, the two experiments above are computationally indistinguishable under DDH and DCR (via straightforward reduction to the semantic security of El-Gamal and Paillier which are equivalent to DDH and DCR respectively). So, to conclude, we will show that the the second experiment is ε-close to the real execution for carefully chosen r, and we will bound the running time of $\mathcal{S}$.

Claim 4.9 (Running Time)**.** *For every $\rho \in \mathsf{poly}$, $\mathcal{S}$ halts in time $O(\rho(\kappa) \cdot n(r + \mathsf{time}_\Sigma))$ with probability at most $e^{-\rho(\kappa)}$ where time_Σ denotes the running time of $\Sigma_{\mathtt{ecdsa}}$.*

Proof. The blowup in the running time of $\mathcal{S}$ compared to time_Σ boils down the rewinding that the simulator performs. Namely, we recall that the simulator rewinds the adversary (i) every time $\mathcal{A}$ decides to corrupt the special party and (ii) during the simulation of the key-generation until $\mathcal{S}$ extracts the relevant data from $\mathcal{A}$ – and $\mathcal{S}$ halts if it fails to do so after r attempts. So, each time the adversary guesses the special party, the simulator spends r "time units" to extract the relevant secrets and then it spends another time_Σ to carry out the rest of the simulation. So, the probability that $\mathcal{S}$'s running time is greater than $\rho(\kappa) \cdot n(r + \mathsf{time}_\Sigma)$ is bounded by $(1 - 1/n)^{n \cdot \rho(\kappa)}$, i.e. the adversary guesses the honest party each and every time. Conclude using the inequality $\log(1 - 1/n) \leq -1/n$. □

Claim 4.10 (ε-Closeness.)**.** *For $r = 2\kappa/\varepsilon \in \mathsf{poly}$, it holds that Experiment 2 is ε-close to the real execution.*

Proof. Write $\boldsymbol{p}$ for the probability that the adversary does not abort (i.e. quits or returns an invalid proof) during the extraction event and notice that $\boldsymbol{p}$ is a random variable which depends on $\mathcal{A}$'s random coins and the transcript so far. Furthermore, for fixed $\mu \leftarrow \boldsymbol{p}$, if $\mu \geq \varepsilon/2$, then the probability that $\mathcal{S}$ fails to extract is at most $e^{-\kappa/8}$ by Chernoff bound (cf. Fact 4.11 below with $d = 1/2$). To conclude, notice that the Experiment 2 is distinguishable from the real execution in one of the following events, and only then. For $\mu \leftarrow \boldsymbol{p}$,

1. $\mu < \varepsilon/2$ and the first run of the protocol does *not* lead to an aborting execution.
2. $\mu \geq \varepsilon/2$ and the simulator fails to extract after r attempts.
3. One of the simulated ZK proofs is distinguishable from the real proof.
4. The adversary breaks soundness in one of ZK proofs.

Item 1 happens with probability at most $\varepsilon/2$, Item 2 happens with probability at most $e^{-\kappa/8}$ and Items 3 and 4 happen with probability at most $\eta \in \mathsf{negl}$, because the output of each ZK simulator is statistically indistinguishable from the real transcript of the proof with $(2^{-\nu+2\kappa})$-closeness and $\nu - 2\kappa \in \omega(\log(\kappa))$, and all the proofs are computationally sound by Theorem 5.1 under strong RSA. Overall, the two experiments are distinguishable with probability at most $\varepsilon/2 + e^{-\kappa/8} + \eta \leq \varepsilon$.

□

Fact 4.11 (Chernoff Bound)**.** *Define $\boldsymbol{\sigma} = \sum_{i=1}^{n} \boldsymbol{y}_i$ for $\boldsymbol{y}_1 \ldots \boldsymbol{y}_r$ iid Boolean variables with $\Pr[\boldsymbol{y}_i = 1] = \mu$. It holds that $\Pr[\boldsymbol{\sigma} \leq (1 - d)r \cdot \mu] \leq e^{-r\mu d^2/2}$, for every $d \geq 0$.*

This concludes the proof of Theorem 4.8. □

5 Proof of Soundness for Batch-Proving

In this section, we analyze the soundness and HVZK of a generic m-batch Schnorr protocol depending on HVZK of the underlying non-batch protocol and some additional technical requirements. Then, in Section 5.2 we show that our batch protocols from Section 3 satisfy the hypothesis of Theorem 5.1 for suitable choice of parameters.

Theorem 5.1. *Let Π denote a non-batch μ-HVZK Schnorr Protocol for tuple $(\phi, \boldsymbol{E}, \boldsymbol{R}, \boldsymbol{S})$ and write Π^* for the associated m-batch protocol. Assume that*

1. *Π^* satisfies $(\varepsilon, \boldsymbol{V})$-extractability for some set $\boldsymbol{V} \subseteq 2^{\boldsymbol{E}^m}$.*
2. *For all $j \leq m$, if $\{\vec{e}_1, \ldots, \vec{e}_j\} \in \boldsymbol{V}$ then $\Pr_{\vec{e}_{j+1} \leftarrow \boldsymbol{E}^m}[\{\vec{e}_1, \ldots, \vec{e}_j, \vec{e}_{j+1}\} \notin \boldsymbol{V}] \leq \beta$.*

3. For all $\vec{w} \notin \boldsymbol{S}^m$, it holds that $\Pr_{\vec{e} \leftarrow \boldsymbol{E}^m}[\alpha + \sum_j e_j w_j \in \boldsymbol{S}] \leq \gamma$, for every $\alpha \in \mathbb{H}$.

4. For $w, w' \leftarrow \mathcal{A}$ such that $w \neq w'$, it holds that $\Pr[\phi(w) = \phi(w')] \leq \delta$, for every efficient $\mathcal{A}$.

We refer to Items 2, 3 and 4 as β-Robustness, γ-Unpredictability and δ-Collision Resistance, resp.

Then, for $\varepsilon, \beta, \gamma, \delta \in \mathsf{negl}(\kappa)$, it holds that Π^* is (μ^*, ν^*)-secure for $\mu^* = m \cdot \mu$ and $\nu^* \in \mathsf{negl}$.

Before we prove Theorem 5.1 we point the reader to the relevant claims for showing that the batch Schnorr protocols from Section 3 are sound.

Claim 5.2 (m-Batch Soundness, Section 3)**.** *Let $(\phi, \boldsymbol{E}, \boldsymbol{R}, \boldsymbol{S})$ denote the underlying tuple of $\Pi \in \{\{\Xi_i\}_{i=1}^n, \Theta'\}$ as per Definitions 3.5 and 3.8. Write Π^* for the m-batch variant and let $\boldsymbol{V}$ denote the set from Definition 5.6. Then, there exists $\varepsilon, \beta, \gamma, \delta \in \mathsf{negl}$ such that*

1. *Under the strong RSA assumption, it holds that Π^* is $(\varepsilon, \boldsymbol{V})$-Extractable over the choice $\pi \leftarrow \mathsf{ped}(1^\kappa)$.*
2. *If $N = p_1 p_2$ is Paillier-Blum s.t. $p_1, p_2 > q$, then $\boldsymbol{V}$ is β-Robust.*
3. *Unconditionally, Π^* is γ-Unpredictable.*
4. *Under the factoring assumption (implied by strong RSA), ϕ is δ-Collision Resistant.*

Proof. $\varepsilon \in \mathsf{negl}(\kappa)$ by Theorem 5.7, and $\beta, \gamma, \delta \in \mathsf{negl}(\kappa)$ by Fact 5.11, Fact 5.12 and Fact 5.13 respectively. □

5.1 Proof of Theorem 5.1

HVZK. First, we show the zero-knowledge property. Write $\boldsymbol{\sigma}_i$ for the random variable calculated as $\boldsymbol{\sigma}_i = \boldsymbol{\alpha} + \sum_{j=1}^i \boldsymbol{e}_j w_j$ where $\boldsymbol{\alpha}$ and $(\boldsymbol{e}_1, \ldots, \boldsymbol{e}_i)$ are sampled from the distributions $\alpha \leftarrow \boldsymbol{E}$ and $\vec{e} \leftarrow \boldsymbol{E}^i$ respectively. Note that $\mathrm{SD}(\boldsymbol{\sigma}_0, \boldsymbol{\sigma}_1) \leq \mu$ by the HVZK property of the underlying protocol. Further observe that $\mathrm{SD}(\boldsymbol{\sigma}_0, \boldsymbol{\sigma}_m) = \mathrm{SD}(\boldsymbol{\sigma}_0, \boldsymbol{\sigma}_{m-1} + \boldsymbol{e}_m w_m) \leq \mathrm{SD}(\boldsymbol{\sigma}_0, \boldsymbol{\sigma}_{m-1}) + \mathrm{SD}(\boldsymbol{\sigma}_0, \boldsymbol{\sigma}_0 + \boldsymbol{e}_m w_m)$ and the HVZK part of the claim follows by simple induction.

Soundness. Hereafter, assume that $\vec{X}$ does not admit a suitable preimage in $\boldsymbol{S}^m$ and let $\mathcal{A}$ denote an adversary that breaks soundness with probability λ, i.e. the probability that $(A, \vec{e}, z)$ is a valid transcript for $\vec{X}$ is at east λ, where A and z are chosen by the adversary for a random $\vec{e} \leftarrow \boldsymbol{E}^m$ (and z may depend on $\vec{e}$). Consider the following experiment.

Experiment 5.3. For $r \in \mathsf{poly}$, define $\mathcal{E}_r^{\mathcal{A}}(\vec{X})$ with black-box access to $\mathcal{A}$ as follows.

Operation.

1. Run the adversary to obtain the first message $A \leftarrow \mathcal{A}(X_1, \ldots, X_m)$ for Π^*.
2. Sample $\vec{e}_1, \ldots, \vec{e}_r \leftarrow \boldsymbol{E}^m$ iid and hand it to $\mathcal{A}$ as the verifier's response (in r parallel executions).
3. Obtain (possibly invalid) transcripts $\tau_1, \ldots, \tau_r$ using the last message(s) of $\mathcal{A}$.

Write $\vec{f}_1, \ldots, \vec{f}_{m+1}$ for the (possibly shorter or empty) subsequence of $\vec{e}_1, \ldots, \vec{e}_r$ consisting of the first $m+1$ vectors $\vec{e}_j$ such that $\mathcal{A}$ returns a valid transcript for $\vec{e}_j$.

Output. If $\{\vec{f}_1, \ldots, \vec{f}_{m+1}\} \notin \boldsymbol{V}$ or there are fewer than $m+2$ accepting transcripts then output 0. Else, output 1 together with the first $m+2$ accepting transcripts.

Claim 5.4. *$\mathcal{E}$ outputs 0 in Experiment 5.3 with probability at most $\beta \cdot rm + e^{-d^2\lambda r/2}$ for $d = 1 - (m+1)/\lambda r$.*

Proof. We recall the Chernoff bound. For $\boldsymbol{y}_1 \ldots \boldsymbol{y}_r$ iid Boolean random variables with $\Pr[\boldsymbol{y}_i = 1] = \mu$ and $\boldsymbol{\sigma} = \sum_{i=1}^n \boldsymbol{y}_i$ it holds that $\Pr[\boldsymbol{\sigma} \leq (1-d)r \cdot \mu] \leq e^{-r\mu d^2/2}$, for every $d \geq 0$. Hence, the probability that $\mathcal{E}$ extracts fewer than $m+1$ valid transcripts in Item 1 of Experiment 5.3 is $e^{-d^2\lambda r/2}$ for $d = 1 - (m+1)/\lambda r$. Furthermore, the probability that $(\vec{f}_1, \ldots, \vec{f}_{m+1}) \notin \boldsymbol{V}$ is at most $\beta \cdot rm$, by union bound. □

Consider the following sequence of events.

1. Run Experiment 5.3 and obtain $m+2$ valid transcripts.
2. Use the first $m+1$ transcripts to compute $\alpha, (w_i)_{i=1}^m$ such that $A = \phi(\alpha)$ and $\phi(w_j) = X_j$.

   Let $\tau^\star = (A, \vec{e}^\star, z^\star)$ denote the last (unused) transcript.
3. $z^\star = \alpha + \sum_{j=1}^m e_j^\star w_j \notin \boldsymbol{S}$.
4. $z^\star \neq \alpha + \sum_{j=1}^m e_j^\star w_j$ and $\phi(z^\star) = \phi(\alpha + \sum_{j=1}^m e_j^\star w_j)$.

In summary, $\mathcal{A}$ breaks soundness only if one of the above does not happen, i.e. with probability at most

$$\underbrace{e^{-r\lambda d^2/2} + \beta \cdot mr + \varepsilon}_{\text{Items 1, 2}} + \underbrace{r \cdot \gamma + \delta}_{\text{Items 3, 4}} \geq \lambda \tag{5}$$

Claim 5.5. *If* $\varepsilon, \beta, \gamma, \delta \in \mathsf{negl}$*, then* $\lambda \in \mathsf{negl}$.

Proof. Suppose that $\lambda \notin \mathsf{negl}$ i.e. there exists $\ell \in \mathsf{poly}$ such that $\lambda \geq 1/\ell$ infinitely often. Fix $r \in \mathsf{poly}$ such $r/\ell \in \omega(\log(\kappa))$ and $m \cdot \ell/r \in o(1)$. By Equation (5), deduce that there exists $\eta \in \mathsf{negl}$ such that $\eta + \beta mr + \varepsilon + r\gamma + \delta \geq 1/\ell$ infinitely often, which yields a contradiction since $\varepsilon, \beta, \gamma, \delta \in \mathsf{negl}$. □

This concludes the proof of Theorem 5.1. □

5.2 Putting Everything Together

In this section we prove auxiliary claims for showing our batch protocols from Section 3 satisfy the hypothesis of Theorem 5.1 for suitable choice of parameters. We first observe that all the protocols can be cast as Schnorr protocols for $(\phi, \boldsymbol{E}, \boldsymbol{R}, \boldsymbol{S})$ such that

$$\begin{aligned} \phi : \mathbb{Z}^r \times \mathbb{F}_q^\alpha \times \mathbb{Z}_{N_1}^{*\beta_1} \times \cdots \times \mathbb{Z}_{N_n}^{*\beta_n} \times \mathbb{Z} \to \mathbb{G}^\gamma \times \mathbb{Z}_{N_1^2}^{*\beta_1} \times \ldots \times \mathbb{Z}_{N_n^2}^{*\beta_n} \times \mathbb{Z}_{\hat{N}}^* \\ (\vec{w}, \vec{\mu}, \vec{\nu}_1, \ldots, \vec{\nu}_n, \rho) \mapsto (\phi_0(\vec{w}, \vec{\mu}), \phi_1(\vec{w}, \vec{\nu}_1), \ldots, \phi_n(\vec{w}, \vec{\nu}_n), \theta(\vec{w}, \rho)) \end{aligned}$$

where

$$\begin{cases} r, \alpha, \beta_1, \ldots, \beta_n, \gamma \in \mathbb{Z} \\ \phi_i(\vec{w}, \vec{\nu}) = (\nu_1^{N_i} \prod_{k=1}^r A_{i,1,k}^{w_j}, \ldots, \nu_{\beta_i}^{N_i} \prod_{k=1}^r A_{i,\beta_i,k}^{w_j}) \in \mathbb{Z}_{N_i^2}^{*\beta_i} \quad \text{for } (A_{i,j,k} \in \mathbb{Z}_{N_i^2}^*)_{j \in [\beta_i], k \in [r]} \\ \theta(\vec{w}, \rho) = t^\rho \prod_{j=1}^r s_j^{w_j} \mod \hat{N} \\ (\vec{w}, \ldots, \rho) \in \boldsymbol{R} \text{ or } \boldsymbol{S} \text{ iff } \vec{w}, \rho \in \boldsymbol{I}(\cdot) \text{ or } \boldsymbol{J}(\cdot) \end{cases}$$

Definition 5.6. Let Π denote the m-batch protocol for the tuple above and define $\boldsymbol{V} \subseteq 2^{\boldsymbol{E}^m}$ such that $\{\vec{e}_1, \ldots, \vec{e}_k\} \in \boldsymbol{V}$ iff there exists $\vec{e}_{k+1} \ldots \vec{e}_{m+1} \in \boldsymbol{E}^m$ such that the matrix below is invertible over $\mathbb{F}_q$ and $(\mathbb{Z}_{N_i}, +, \cdot)$, for all $i \in [n]$.

$$E = \begin{pmatrix} \vec{e}_1 & 1 \\ \vdots & \vdots \\ \vec{e}_{m+1} & 1 \end{pmatrix} = \begin{pmatrix} e_{1,1} & \ldots & e_{1,m} & 1 \\ \vdots & \vdots & & \vdots \\ e_{m+1,1} & \ldots & e_{m+1,m} & 1 \end{pmatrix} \tag{6}$$

In the remainder, we state and prove Theorem 5.7 (Extractability) and Facts 5.11 (Robustness), 5.12 (Unpredictability) and 5.13 (Collision Resistance).

5.2.1 Extractability

Theorem 5.7. *The following holds under the strong-RSA assumption. For $(\hat{N}, t, s_1, \ldots) \leftarrow \mathsf{ped}(1^\kappa)$, letting $\boldsymbol{V}$ be as above, there exists $\varepsilon \in \mathsf{negl}$ such that Π is $(\varepsilon, \boldsymbol{V})$-extractable.*

Proof. For strong-RSA challenge $(N, c) \leftarrow \mathsf{sRSA}(1^\kappa)$, the Pedersen parameters $(\hat{N}, t, s_1, \ldots)$ are set as[18] $(\hat{N}, t) = (N, c)$ and $s_k = t^{\lambda_k} \mod \hat{N}$ for $\lambda_k \leftarrow [\hat{N}^2]$ and let $Q = |\langle t \rangle|$ denote the size of the group generated by $t \in \mathbb{Z}_N^*$.[19] We will show a reduction from Extractability to strong RSA; we will be using the λ's in the reduction. We prove the claim for ϕ such that[20]

$$\begin{aligned} \phi : \mathbb{Z}^r \times \mathbb{Z}_{N_0}^* \times \mathbb{Z} \to \mathbb{Z}_{N_0^2}^* \times \mathbb{Z}_{\hat{N}}^* \\ (\vec{w}, \nu, \rho) \mapsto (\nu^{N_0} \prod_{k=1}^{r} A_k^{w_k}, t^\rho \prod_{k=1}^{r} s_k^{w_k}) \end{aligned}$$

So, for $(\vec{C}, \vec{S}) \in (\mathbb{Z}_{N_0^2} \times \mathbb{Z}_{\hat{N}})^m$, let $\{\tau_i = ((D, T), \vec{e}_i, (\vec{z}_i, \mu_i, \gamma_i))\}_{i \in [m+1]}$ denote $m+1$ valid transcripts i.e.

$$\forall i, \quad \begin{cases} \mu_i^{N_0} \prod_{k=1}^{r} A_k^{z_{i,k}} = D \cdot \prod_{j=1}^{m} C_j^{e_{i,j}} \mod N_0 \\ t^{\gamma_i} \prod_{k=1}^{r} s_k^{z_{i,k}} = T \cdot \prod_{j=1}^{m} S_j^{e_{i,j}} \mod \hat{N} \end{cases} \tag{7}$$

and assume that $\{\vec{e}_i\}_{i=1}^{m+1} \in \boldsymbol{V}$. Define the square integer matrix E as in Equation (6) and let $\Delta_e = \det(E)$. By assumption, Δ_e is invertible over $\mathbb{Z}_{N_0}$, thus also over $\mathbb{Q}$, and there exists an integer matrix E^* satisfying $E^* \cdot E = \Delta_e \cdot \mathsf{id}$. Fix $i \in [m+1]$ and define $\Delta_\gamma = \sum_{j=1}^{m+1} e_{i,j}^* \cdot \gamma_j$ and $\Delta_z^{(k)} = \sum_{j=1}^{m+1} e_{i,j}^* \cdot z_{j,k}$, where $e_{i,j}^*$ is the entry of E^* indexed by (i,j). Observe that

$$t^{\Delta_\gamma} \cdot \prod_{k=1}^{r} s_k^{\Delta_z^{(k)}} = R^{\Delta_e} \mod \hat{N} \text{ s.t. } \begin{cases} R = S_i & \text{if } i \neq m+1 \\ R = T & \text{otherwise} \end{cases} \tag{8}$$

So, notice that if Δ_e divides Δ_γ and $\{\Delta_z^{(k)}\}_{k=1}^r$ over the integers, then at least one of the following is true

1. $\Delta_e \neq 1$ and $\gcd(\Delta_e, Q) \neq 1$.
2. $t^{\Delta_\gamma / \Delta_e} \cdot \prod_{k=1}^{r} s_k^{\Delta_z^{(k)} / \Delta_e} = R \mod \hat{N}$.

Item 1 yields the factorization[21] of $\hat{N}$ and thus it suffices to bound the probability that Δ_e does not divide one of $\{\Delta_z^{(k)}\}_{k=1}^r$ or Δ_γ. Let $\hat{\varepsilon} = \Pr[\Delta_e \nmid \Delta_\gamma \ \vee \ \Delta_e \nmid \Delta_z^{(1)} \ \vee \ \ldots \ \vee \ \Delta_e \nmid \Delta_z^{(r)}]$ where the probability is calculated over the prover's coins and the choice of $(\hat{N}, t, s_1, \ldots)$. Further define $\Delta_\Sigma = \Delta_\gamma + \sum_k \lambda_k \Delta_z^{(k)}$ and observe that $\hat{\varepsilon} \leq \Pr[\Delta_e \nmid \Delta_\Sigma] + \sum_{k=1}^{r} \Pr[\Delta_e \nmid \Delta_z^{(k)} \ \wedge \ \Delta_e \mid \Delta_\Sigma]$. Apply Claim 5.8, Claim 5.9 and Claim 5.10 and conclude that, since $i \in [m+1]$ was chosen arbitrarily,

$$\varepsilon \leq (m+1) \cdot \hat{\varepsilon} + \mathsf{negl}(\kappa) \in \mathsf{negl}(\kappa). \tag{9}$$

Claim 5.8. *It holds that* $\Pr[\Delta_e \nmid \Delta_\Sigma] \in \mathsf{negl}(\kappa)$,

Proof. Let $d = \gcd(\Delta_e, \Delta_\Sigma)$. If $\Delta_e \nmid \Delta_\Sigma$, then at least one the following is true.

1. $d \neq 1$ and $\gcd(d, Q) \neq 1$.
2. $t^{d_\Sigma} = R^{d_e} \mod \hat{N}$ for $d \cdot d_e = \Delta_e$ and $d \cdot d_\Sigma = \Delta_\Sigma$.

For Item 2, let (u, v) denote the Bézout coefficients of (d_e, d_Σ) i.e. $u \cdot d_e + v \cdot d_\Sigma = 1$, and deduce $m = t^u R^v \mod \hat{N}$ and d_e solve strong RSA since $t = t^{u \cdot d_e + v \cdot d_\Sigma} = t^{u \cdot d_e} \cdot R^{v \cdot d_e} = (t^u \cdot R^v)^{d_e} = m^{d_e} \mod N$. ☐

[18] This is the right distribution for the Pedersen parameters

[19] We recall that $Q = \varphi(N)/4$ is a biprime where φ is the Euler function (with overwhelming probability).

[20] The general case follows straightforwardly.

[21] Factoring is reducible to strong RSA.

Next, we prove that if $\Delta_e \nmid \Delta_z^{(j)}$ for some $j \in r$, then the probability that $\Delta_e \mid \Delta_\Sigma$ is bounded away from 1 (together with Claim 5.8, this yields Equation (9)).

Claim 5.9. $\Pr\left[\Delta_e \mid \Delta_\Sigma \;\middle|\; \Delta_e \nmid \Delta_z^{(j)}\right] \leq 1/2 + \mathsf{negl}(\kappa)$, *for every* j.

Proof. Define $\hat{\Delta}_j = \Delta_\gamma + \sum_{k \neq j} \lambda_k \Delta_z^{(k)}$ and write $\Delta_\Sigma = \hat{\Delta}_j + \hat{\lambda}_j \cdot \Delta_z^{(j)} + \rho_j \cdot Q \cdot \Delta_z^{(j)}$ where $\hat{\lambda}_j = \lambda_j \mod Q$ and ρ_j is uniquely determined. We make the following preliminary observations. If $\Delta_e \nmid Q \cdot \Delta_z^{(j)}$, then there exists a prime power a^b such that

$$\begin{cases} a^b \mid Q \cdot \Delta_z^{(j)} \\ a^{b+1} \nmid Q \cdot \Delta_z^{(j)} \\ a^{b+1} \mid \Delta_e \end{cases}$$

Finally, notice that if $\Delta_e \nmid Q \cdot \Delta_z^{(j)}$ and $\Delta_e \mid \Delta_\Sigma$, then, using the notation above,

$$\hat{\Delta}_j + \hat{\lambda}_j \cdot \Delta_z^{(j)} + \rho_j \cdot Q \cdot \Delta_z^{(j)} = 0 \mod a^b$$

and thus ρ_j is uniquely determined modulo a. Thus,

$$\Pr\left[\Delta_e \mid \Delta_\Sigma \;\middle|\; \Delta_e \nmid \Delta_z^{(j)}\right] \leq \Pr[\gcd(\Delta_e, Q) \neq 1] + \Pr\left[\Delta_e \mid \Delta_\Sigma \;\middle|\; \Delta_e \nmid \Delta_z^{(j)} \wedge \gcd(\Delta_e, Q) = 1\right]$$

and $\Pr\left[\Delta_e \mid \Delta_\Sigma \;\middle|\; \Delta_e \nmid \Delta_z^{(j)} \wedge \gcd(\Delta_e, Q) = 1\right] \leq 1/a$, which concludes the proof of the claim. ☐

Next, set $\vec{w}_i, \vec{\alpha} \in \mathbb{Z}^r$ and $\rho_i, \eta \in \mathbb{Z}$ such that $w_{i,k} = (\sum_j e^*_{i,j} z_{j,k})/\Delta_e$ and $\rho_i = (\sum_j e^*_{i,j} \gamma_j)/\Delta_e$ and $\alpha_k = (\sum_j e^*_{m+1,j} z_{j,k})/\Delta_e$ and $\eta = (\sum_j e^*_{m+1,j} \gamma_j)/\Delta_e$, and it remains to show that the $\vec{w}_i$'s are the preimages of the C_i's and that we can extract the randomizers (the ν's). From Equation (7) , since Δ_e is coprime to N_0 (by assumption, since Δ_e is invertible in $\mathbb{Z}_{N_0}$), deduce that

$$\left(\prod_{j=1}^{m} \mu_j^{e^*_{i,j}}\right)^{N_0} = \left(B \cdot \prod_{k=1}^{r} A_k^{-w_{i,k}}\right)^{\Delta_e} \mod N_0^2 \text{ s.t. } \begin{cases} B = C_i & \text{if } i \neq m+1 \\ B = D & \text{otherwise} \end{cases}$$

and use Claim 5.10 to extract the randomizers (the ν's).

Claim 5.10. *Suppose that* $y^N = x^k \mod p$, *where* k *and* N *are coprime and* $x, y \in \mathbb{Z}_p^*$. *Then, there exists* $\alpha \in \mathbb{Z}_p^*$ *such that* $\alpha^N = x \mod p$. *Furthermore,* α *can be computed efficiently as a function* $|p|$.

Proof. Since k and N are coprime, there exists $u, v \in \mathbb{Z}$ such that $ku + Nv = 1$. Thus $x^{ku+Nv} = x$, and consequently $(y^u \cdot x^v)^N = x^{ku} \cdot (x^N)^v = x \mod p$. For the penultimate equality, notice that y^u and x^v are well defined in $\mathbb{Z}_p^*$. ☐

This concludes the proof Theorem 5.7. ☐

5.2.2 Robustness, Unpredictability and Collision Resistance

Recall that $\boldsymbol{I}(M) = \pm 2^\kappa$ and $\boldsymbol{J}(M) = \pm M \cdot 2^\nu$ where ν is chosen such that such that $\nu - 2\kappa \in \omega(\log(\kappa))$.

Fact 5.11 (Robustness)**.** *Let* $p \in \mathbb{Z}$ *such that* $\log(p) \geq \kappa$ *and assume that* p *is prime. If* $(\vec{e}_1, 1) \ldots (\vec{e}_j, 1) \in \mathbb{Z}^{m+1}$ *are linearly independent over* $\mathbb{Z}_p$ *then*

$$\Pr_{\vec{e}_{j+1} \leftarrow \boldsymbol{I}(2^\kappa)^m}[(\vec{e}_{j+1}, 1) \in \langle (\vec{e}_1, 1), \ldots, (\vec{e}_j, 1) \rangle_{\mathbb{Z}_p}] \leq 1/2^{\kappa-1}.$$

Proof. Let $\vec{v} \in \mathbb{Z}_p^{m+1}$ be an (arbitrary) vector in the orthogonal complement of $\langle (\vec{e}_1, 1), \ldots, (\vec{e}_j, 1) \rangle_{\mathbb{Z}_p}$. Clearly $\Pr[\vec{v} \cdot (\vec{e}_{j+1}, 1) = 0] \leq 2/p$, where $\vec{v} \cdot \vec{u}$ denotes the inner product of $\vec{v}$ and $\vec{u}$ in $\mathbb{Z}_p^{m+1}$. ☐

Fact 5.12 (Unpredictability)**.** *For any $\alpha \in \mathbb{Z}$ and $K \in \mathbb{N}$, if $\vec{w} \notin \boldsymbol{J}(K)^m$ then*

$$\Pr_{\vec{e} \leftarrow \boldsymbol{I}(K)^m}[\alpha + \sum_j e_j w_j \in \boldsymbol{J}(K)] \leq 3/2^\kappa.$$

Proof. For any fixed $\alpha \in \mathbb{Z}$, for any $|w| > K \cdot 2^\nu$,

$$\Pr_{e \leftarrow \boldsymbol{I}(2^\kappa)}[\alpha + we \in \pm K \cdot 2^\nu] = \Pr[ew \in -\alpha \pm K \cdot 2^\nu] \leq \Pr[e \in \lceil -\alpha/w \rfloor \pm 1] \leq \frac{3}{2^\kappa}.$$

□

Fact 5.13 (Collision Resistance – Pedersen Binding)**.** *The following holds under the factoring assumption, for every PPTM $\mathcal{A}$. For $\pi = (\hat{N}, t, \vec{s}, \ldots) \leftarrow \mathsf{ped}(1^\kappa)$, it holds that*

$$\Pr\left[\vec{w} \neq \vec{x} \leftarrow \mathcal{A}(\pi) \ \ s.t. \ \ t^{w_0} \prod_{i=1}^{r} s_i^{w_i} = t^{x_0} \prod_{i=1}^{r} s_i^{x_i} \mod \hat{N}\right] \in \mathsf{negl}(\kappa).$$

Proof. Similarly to the proof of Theorem 5.7, we will use $\mathcal{A}$ to break factoring. Assuming that $s_i = t^{\lambda_i} \mod \hat{N}$ for $\lambda_i \leftarrow [\hat{N}^2]$ with $\lambda_0 = 1$, and write $Q = |\langle t \rangle|$. Let $\Delta = \sum_{i=1}^{r} \lambda_i (w_i - x_i)$ and note that $t^\Delta = 1 \mod \hat{N}$. So, either

1. Q divides Δ (which yields the factorization of $\hat{N}$).
2. or $\Delta = 0$.

To conclude the proof, we show that the probability (over the λ's) of Item 2 is bounded away from 1. Fix i such that $x_i - w_i \neq 0$ and let $\hat{\lambda}, \rho$ such that $\lambda_i = \hat{\lambda} + Q\rho$ and $\hat{\lambda} = \lambda_i \mod Q$. For $z = \sum_{j \neq i} \lambda_j (w_j - x_j)$, deduce that $\rho = \frac{1}{Q} \cdot \left(\frac{-z}{w_i - x_i} - \hat{\lambda}\right)$ which happens with negligible probability (since ρ is info-theoretically hidden). □

References

[1] D. Abram, A. Nof, C. Orlandi, P. Scholl, and O. Shlomovits. Low-bandwidth threshold ECDSA via pseudorandom correlation generators. In *IEEE Symposium on Security and Privacy*, pages 2554–2572. IEEE Computer Society Press, May 2022. doi: 10.1109/SP46214.2022.9833559.

[2] J.-P. Aumasson, A. Hamelink, and O. Shlomovits. A survey of ECDSA threshold signing. Cryptology ePrint Archive, Report 2020/1390, 2020. https://eprint.iacr.org/2020/1390.

[3] E. Bangerter. *Efficient zero knowledge proofs of knowledge for homomorphisms*. PhD thesis, Ruhr University Bochum, 2005.

[4] D. Boneh, B. Bünz, and B. Fisch. Batching techniques for accumulators with applications to IOPs and stateless blockchains. In A. Boldyreva and D. Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 561–586. Springer, Heidelberg, Aug. 2019. doi: 10.1007/978-3-030-26948-7_20.

[5] 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, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 280–312. Springer, Heidelberg, Apr. / May 2018. doi: 10.1007/978-3-319-78381-9_11.

[6] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, Oct. 2001. doi: 10.1109/SFCS.2001.959888.

[7] R. Canetti. Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Report 2003/239, 2003. https://eprint.iacr.org/2003/239.

[8] R. Canetti, A. Jain, and A. Scafuro. Practical UC security with a global random oracle. In G.-J. Ahn, M. Yung, and N. Li, editors, *ACM CCS 2014*, pages 597–608. ACM Press, Nov. 2014. doi: 10.1145/2660267.2660374.

[9] 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, *ACM CCS 2020*, pages 1769–1787. ACM Press, Nov. 2020. doi: 10.1145/3372297.3423367.

[10] 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.

[11] 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.

[12] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker. Two-party ECDSA from hash proof systems and efficient instantiations. In A. Boldyreva and D. Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 191–221. Springer, Heidelberg, Aug. 2019. doi: 10.1007/978-3-030-26954-8_7.

[13] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker. Bandwidth-efficient threshold EC-DSA. In A. Kiayias, M. Kohlweiss, P. Wallden, and V. Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 266–296. Springer, Heidelberg, May 2020. doi: 10.1007/978-3-030-45388-6_10.

[14] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker. Bandwidth-efficient threshold EC-DSA revisited: Online/offline extensions, identifiable aborts, proactivity and adaptive security. Cryptology ePrint Archive, Report 2021/291, 2021. https://eprint.iacr.org/2021/291.

[15] G. Couteau, T. Peters, and D. Pointcheval. Removing the strong RSA assumption from arguments over the integers. In J.-S. Coron and J. B. Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 321–350. Springer, Heidelberg, Apr. / May 2017. doi: 10.1007/978-3-319-56614-6_11.

[16] A. P. K. Dalskov, C. Orlandi, M. Keller, K. Shrishak, and H. Shulman. Securing DNSSEC keys via threshold ECDSA from generic MPC. In L. Chen, N. Li, K. Liang, and S. A. Schneider, editors, *ESORICS 2020, Part II*, volume 12309 of *LNCS*, pages 654–673. Springer, Heidelberg, Sept. 2020. doi: 10.1007/978-3-030-59013-0_32.

[17] 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.

[18] Y. Desmedt. Society and group oriented cryptography: A new concept. In C. Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 120–127. Springer, Heidelberg, Aug. 1988. doi: 10.1007/3-540-48184-2_8.

[19] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In G. Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 307–315. Springer, Heidelberg, Aug. 1990. doi: 10.1007/0-387-34805-0_28.

[20] J. Doerner, Y. Kondi, E. Lee, and a. shelat. Threshold ECDSA from ECDSA assumptions: The multiparty case. In *2019 IEEE Symposium on Security and Privacy*, pages 1051–1066. IEEE Computer Society Press, May 2019. doi: 10.1109/SP.2019.00024.

[21] M. Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In V. Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 152–168. Springer, Heidelberg, Aug. 2005. doi: 10.1007/11535218_10.

[22] E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In B. S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 16–30. Springer, Heidelberg, Aug. 1997. doi: 10.1007/BFb0052225.

[23] A. Gągol, J. Kula, D. Straszak, and M. Świętek. Threshold ECDSA for decentralized asset custody. Cryptology ePrint Archive, Report 2020/498, 2020. https://eprint.iacr.org/2020/498.

[24] 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.

[25] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. *Inf. Comput.*, 164(1):54–84, 2001.

[26] R. Gennaro, D. Leigh, R. Sundaram, and W. S. Yerazunis. Batching Schnorr identification scheme with applications to privacy-preserving authorization and low-bandwidth communication devices. In P. J. Lee, editor, *ASIACRYPT 2004*, volume 3329 of *LNCS*, pages 276–292. Springer, Heidelberg, Dec. 2004. doi: 10.1007/978-3-540-30539-2_20.

[27] J. Groth and V. Shoup. On the security of ECDSA with additive key derivation and presignatures. In O. Dunkelman and S. Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 365–396. Springer, Heidelberg, May / June 2022. doi: 10.1007/978-3-031-06944-4_13.

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

[29] S. Krenn and M. Orrù. Proposal : Sigma-protocols. 2021.

[30] Y. Lindell. Fast secure two-party ECDSA signing. In J. Katz and H. Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 613–644. Springer, Heidelberg, Aug. 2017. doi: 10.1007/978-3-319-63715-0_21.

[31] Y. Lindell and A. Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *ACM CCS 2018*, pages 1837–1854. ACM Press, Oct. 2018. doi: 10.1145/3243734.3243788.

[32] P. D. MacKenzie and M. K. Reiter. Two-party generation of DSA signatures. In J. Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 137–154. Springer, Heidelberg, Aug. 2001. doi: 10.1007/3-540-44647-8_8.

[33] N. Makriyannis. On the classic protocol for mpc schnorr signatures. Cryptology ePrint Archive, Paper 2022/1332, 2022.

[34] N. Makriyannis and U. Peled. A note on the security of gg18. 2021.

[35] U. Maurer. Zero-knowledge proofs of knowledge for group homomorphisms. *Des. Codes Cryptogr.*, 77 (2-3):663–676, 2015.

[36] S. Micali, K. Ohta, and L. Reyzin. Accountable-subgroup multisignatures: Extended abstract. In M. K. Reiter and P. Samarati, editors, *ACM CCS 2001*, pages 245–254. ACM Press, Nov. 2001. doi: 10.1145/501983.502017.

[37] National Institute of Standards and Technology. Digital signature standard (dss). Federal Information Processing Publication 186-4, 2013.

[38] A. Nicolosi, M. N. Krohn, Y. Dodis, and D. Mazières. Proactive two-party signatures for user authentication. In *NDSS 2003*. The Internet Society, Feb. 2003.

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

[40] D. R. Stinson and R. Strobl. Provably secure distributed Schnorr signatures and a (t, n) threshold scheme for implicit certificates. In V. Varadharajan and Y. Mu, editors, *ACISP 01*, volume 2119 of *LNCS*, pages 417–434. Springer, Heidelberg, July 2001. doi: 10.1007/3-540-47719-5_33.

[41] S. A. K. Thyagarajan, A. Bhat, G. Malavolta, N. Döttling, A. Kate, and D. Schröder. Verifiable timed signatures made practical. In J. Ligatti, X. Ou, J. Katz, and G. Vigna, editors, *ACM CCS 2020*, pages 1733–1750. ACM Press, Nov. 2020. doi: 10.1145/3372297.3417263.

[42] D. Tymokhanov and O. Shlomovits. Alpha-rays: Key extraction attacks on threshold ECDSA implementations. *IACR Cryptol. ePrint Arch.*, page 1621, 2021. URL https://eprint.iacr.org/2021/1621.

[43] H. Xue, M. H. Au, X. Xie, T. H. Yuen, and H. Cui. Efficient online-friendly two-party ECDSA signature. In G. Vigna and E. Shi, editors, *ACM CCS 2021*, pages 558–573. ACM Press, Nov. 2021. doi: 10.1145/3460120.3484803.

FIGURE 14 (Ideal Threshold-Signatures Functionality $\mathcal{F}_{\mathsf{tsig}}$)

Key-generation:

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

Signing:

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

Verification:

Upon receiving $(\texttt{sig-vrfy}, \mathit{sid}, m, \sigma, X)$ from a party $\mathcal{X}$, do:

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

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

Key-Refresh:

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

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

Corruption/Decorruption:

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

Figure 14: Ideal Threshold-Signatures Functionality $\mathcal{F}_{\mathsf{tsig}}$

A Realizing `init-tecdsa` via CMP

Party $\mathcal{P}_i$		Party $\mathcal{P}_j$
$\gamma_i \leftarrow \mathbb{F}_q$, $\vec{\Gamma}_i = \mathsf{com}(\gamma_i, Y)$		
$K_i = \mathsf{enc}_i(k_i)$ and $G_i = \mathsf{enc}_i(\gamma_i)$		
$\forall j, \psi_{j,i} \leftarrow (\Xi^1_{j,i})^{\mathsf{FS}}((K_i, \vec{B}_i, G_i, \vec{\Gamma}_i); (k_i, \gamma_i))$		
	$\xrightarrow{\psi_{j,i}, K_i, G_i, \vec{\Gamma}_i}$ $\xleftarrow[\psi_{i,j}, K_j, G_j, \vec{\Gamma}_j]{}$	
$\forall j$, sample $\beta_{i,j}, \hat{\beta}_{i,j} \leftarrow \boldsymbol{I}(2^{2\kappa+\nu})$ and do:		
Set $\hat{D}_{j,i} = [K_j^{\gamma_i} \cdot \mathsf{enc}_j(\hat{\beta}_{i,j})]_{N_j^2}$ and $\hat{F}_{i,j} = \mathsf{enc}_i(\hat{\beta}_{i,j})$		
$D_{j,i} = [K_j^{x_i} \cdot \mathsf{enc}_j(\beta_{i,j})]_{N_j^2}$ and $F_{i,j} = \mathsf{enc}_i(\beta_{i,j})$ and $H_i = H^{\gamma_i}$		
$\phi_{j,i} \leftarrow (\Xi^2_{j,i})^{\mathsf{FS}}((H_i, X_i, \vec{\Gamma}_i, D_{j,i}, F_{i,j}, \hat{D}_{j,i}, \hat{F}_{i,j}); (x_i, \gamma_i, \beta_{i,j}, \hat{\beta}_{i,j}))$		
	$\xrightarrow{H_i, D_{j,i}, F_{i,j}, \hat{D}_{j,i}, \hat{F}_{i,j}, \phi_{j,i}}$ $\xleftarrow[H_j, D_{i,j}, F_{j,i}, \hat{D}_{i,j}, \hat{F}_{j,i}, \phi_{i,j}]{}$	
Set $\delta_i = [k_i\gamma_i + \sum_{j \neq i} \mathsf{dec}_i(\hat{D}_{i,j}) - \hat{\beta}_{i,j}]_q$		
Set $\Lambda = \prod_j H_j$ and $\Delta_i = \Lambda^{k_i}$		
$\omega_i \leftarrow (\Xi^3_\Lambda)^{\mathsf{FS}}(\Delta_i, \vec{B}_i; k_i)$		
	$\xrightarrow{\delta_i, \Delta_i, \omega_i}$ $\xleftarrow[\delta_j, \Delta_j, \omega_j]{}$	
Set $\delta = \sum_j \delta_j$, $\Delta = \prod_j \Delta_j$ and check $g^\delta = \Delta$		
Set $R = H^{\delta^{-1}}$ and $\chi_i = [k_i x_i + \sum_{j \neq i} \mathsf{dec}_i(D_{i,j}) - \beta_{i,j}]_q$		
Set $r = R|_{\text{x-axis}}$ and output (R, χ_i, r)		

Figure 15: Threshold ECDSA, initialize Signing. Protocol of Canetti et al. [11], aka CMP.

Definition A.1. For $(\hat{N}, t, s_1, \ldots) = \pi_j$ and $N = N_i$, define Schnorr protocol $\Xi^1_{j,i}$ for $(\phi, \boldsymbol{E}, \boldsymbol{R}, \boldsymbol{S})$ where $\boldsymbol{E} = \boldsymbol{I}(2^\kappa)$

$$\begin{aligned}\phi : \mathbb{Z}^2 \times \mathbb{Z}_N^{*2} \times \mathbb{F}_q^2 \times \mathbb{Z} &\to (\mathbb{G}^2 \times \mathbb{Z}^*_{N^2})^2 \times \mathbb{Z}^*_{\hat{N}} \\ (k, \gamma, \rho, \lambda, \mu, \nu, \alpha) &\mapsto (\phi_0(k, \mu), \phi_1(k, \rho), \phi_0(\gamma, \nu), \phi_1(\gamma, \lambda), \phi_2(\alpha, k, \gamma))\end{aligned}$$

and

$$\begin{cases}\phi_0 : (k, \mu) \mapsto (g^\mu, Y^\mu g^k) \\ \phi_1 : (k, \rho) \mapsto (1+N)^k \cdot \rho^N \\ \phi_2 : (\alpha, k, \gamma) \mapsto t^\alpha \cdot s_1^k \cdot s_2^\gamma\end{cases}$$

and $(k, \gamma, \ldots, \alpha) \in \boldsymbol{R} \iff k, \gamma \in \boldsymbol{I}(2^\kappa) \wedge \alpha \in \boldsymbol{I}(\hat{N} \cdot 2^\kappa)$. Define $\boldsymbol{S}$ analogously with $\boldsymbol{J}(\cdot)$.

Definition A.2. For $(\hat{N}, t, s_1, \ldots) = \pi_j$ and $(N, M) = (N_j, N_i)$, define Schnorr protocol $\Xi^2_{j,i}$ for $(\phi, \boldsymbol{E}, \boldsymbol{R}, \boldsymbol{S})$ where $\boldsymbol{E} = \boldsymbol{I}(2^\kappa)$

$$\begin{aligned}\phi : \mathbb{Z}^4 \times \mathbb{F}_q \times \mathbb{Z}_N^{*2} \times \mathbb{Z}_M^{*2} \times \mathbb{Z} &\to \mathbb{G} \times \mathbb{G} \times \mathbb{G}^2 \times (\mathbb{Z}^*_{N^2} \times \mathbb{Z}^*_{M^2})^2 \times \mathbb{Z}^*_{\hat{N}} \\ (x, \gamma, \beta, \hat{\beta}, \mu, \nu, \hat{\nu}, \rho, \hat{\rho}, \lambda) &\mapsto (H^\gamma, g^x, \phi_0(\gamma, \mu), \phi_1(x, \beta, \nu, \rho), \phi_1(\gamma, \hat{\beta}, \hat{\nu}, \hat{\rho}), \phi_2(\lambda, x, \gamma, \beta, \hat{\beta}))\end{aligned}$$

and

$$\begin{cases} \phi_0 : (\gamma, \mu) \mapsto (g^{\mu}, Y^{\mu} g^{\gamma}) \\ \phi_1 : (x, \beta, \nu, \rho) \mapsto (K^x \cdot (1+N)^{\beta} \cdot \nu^N, (1+M)^{\beta} \cdot \rho^M) \\ \phi_2 : (\lambda, x, \gamma, \beta, \hat{\beta}) \mapsto t^{\lambda} \cdot s_1^x \cdot s_2^{\gamma} \cdot s_3^{\beta} \cdot s_4^{\hat{\beta}} \end{cases}$$

and

$$\begin{cases} (x, \gamma, \beta, \hat{\beta}, \ldots, \lambda) \in \boldsymbol{R} \iff x, \gamma \in \boldsymbol{I}(2^{\kappa}) \ \wedge \ \beta, \hat{\beta} \in \boldsymbol{I}(2^{2\kappa+\nu}) \ \wedge \ \lambda \in \boldsymbol{I}(\hat{N} \cdot 2^{\kappa}) \\ (x, \gamma, \beta, \hat{\beta}, \ldots, \lambda) \in \boldsymbol{S} \iff x, \gamma \in \boldsymbol{J}(2^{\kappa}) \ \wedge \ \beta, \hat{\beta} \in \boldsymbol{J}(2^{2\kappa+\nu}) \ \wedge \ \lambda \in \boldsymbol{J}(\hat{N} \cdot 2^{\kappa}) \end{cases}$$

Definition A.3. Define Schnorr protocol Ξ^3_{Λ} for $(\phi, \boldsymbol{E})$ where $\boldsymbol{E} = \mathbb{F}_q$ and

$$\phi : \mathbb{F}_q^3 \to \mathbb{G} \times \mathbb{G}^2$$
$$(k, b, v) \mapsto (\Lambda^k, g^b, Y^b \cdot g^k)$$

A.1 Simulator for CMP

FIGURE 16 (Simulator for $\Sigma_{\mathtt{cmp}}$)

Parameters. Adversary $\mathcal{A}$, RO $\mathcal{H}$ and nonce $R \in \mathbb{G}$.

Operation.

Round 1. Set $K_b, G_b = \mathsf{enc}_b(0)$ and $\vec{\Gamma}_b \leftarrow \mathbb{G}^2$.

1. Calculate $(\psi_{j,b})_{j \neq b}$ using the HVZK simulator

   Hand over $((\psi_{j,b})_{j \neq b}, K_b, G_b, \vec{\Gamma}_b)$ to $\mathcal{A}$.

Round 2. When obtaining $(\psi_{b,j}, K_j, G_j, \vec{\Gamma}_j)$, extract k_j, γ_j and do:

1. Set $\{F_{b,j}, \hat{F}_{b,j} = \mathsf{enc}_b(0)\}_{j \neq b}$ and $\{D_{b,j} = \mathsf{enc}_j(\alpha_j), \hat{D}_{b,j} = \mathsf{enc}_j(\hat{\alpha}_j)\}_{j \neq b}$ for $\alpha_{j,b}, \hat{\alpha}_{j,b} \in \boldsymbol{I}(2^{2\kappa+\nu})$
2. Sample $\delta \leftarrow \mathbb{F}_q$ and set $H_b = R^{\delta} \cdot \prod_{j \neq b} H^{-\gamma_j}$
3. Calculate $(\phi_{j,b})_{j \neq b}$ using the HVZK simulator

   Hand over $(H_b, \ldots, \phi_{j,b})_{j \neq b}$ to $\mathcal{A}$.

Round 3. When obtaining $(H_j, \ldots, \phi_{b,j})_{j \neq b}$, γ_j and do:

1. Extract $\{\beta_j = \mathsf{dec}_j(F_{j,b}), \beta_j = \mathsf{dec}_j(\hat{F}_{j,b})\}_{j \neq b}$ using $\mathcal{P}_j$'s secret key.
2. Set $\delta_b = [\delta - \sum_{j \neq b} (\hat{\alpha}_j - \hat{\beta}_j) - (\sum_{i,j \neq b} k_i \gamma_j)]_q$ and and $\Delta_b = g^{\delta} \cdot \prod_{j \neq b} \Lambda^{-k_i}$.
3. Calculate ω_b using the HVZK simulator

   Hand over $(\delta_b, \Delta_b, \omega_b)$ to $\mathcal{A}$.

Output. When obtaining $(\delta_j, \Delta_j, \omega_j)_{j \neq b}$, do:

Output $\chi^* = [\sum_{j \neq b} (\alpha_j - \beta_j) + (\sum_{i,j \neq b} k_i x_j)]$

Figure 16: Simulator for $\Sigma_{\mathtt{cmp}}$

B Missing ZK Protocols/Proofs

B.1 Security Proof for Multi-Pedersen Membership

Recall (Definition 3.1) the m-batch Schnorr protocol Π^* for $(\phi, \boldsymbol{E})$ where $\boldsymbol{E} = \{0, 1\}$ and

$$\phi : \mathbb{Z}_{\varphi(\hat{N})} \to \mathbb{Z}^*_{\hat{N}}$$
$$\alpha \mapsto t^{\alpha}$$

Claim B.1. *It holds that Π^* is (μ, ν)-secure for $\mu = 1 - \frac{\varphi(N)}{N}$ and $\nu = \frac{1}{2}$.*

Proof. Let $\vec{s} = (s_1, \ldots, s_m)$ denote the common input. The HVZK part of the claim follows since (A, e, z) is $(1 - \frac{\varphi(N)}{N})$-close to a honest transcript, for $z \leftarrow [0, N-1]$, $\vec{e} \leftarrow \{0,1\}^m$ and $A = t^z \cdot \prod_{i=1}^m s_i^{-e_i} \mod \hat{N}$. For the soundness part of the claim, we assume the following. WLOG $s_i \notin \langle t \rangle$ for all $i \in [m]$ (otherwise remove all the good s's and consider the smaller batch). Next, let $\vec{e}, \vec{f} \in \{0,1\}^m$ be two Boolean vectors of hamming distance 1 from each other, i.e. $e_i = f_i$ if and only if $i \neq j$, for some $j \in [m]$. Observe that

1. If $t^z = A \cdot \prod_{i=1}^m s_i^{e_i} \mod \hat{N}$ and $t^{z'} = A \cdot \prod_{i=1}^m s_i^{f_i} \mod \hat{N}$ then $A \in \mathbb{Z}_{\hat{N}}^*$, $s_j \in \langle t \rangle$.
2. Any subset of $\{0,1\}^m$ larger than 2^{m-1} contains two vectors that are 1-far from each other.

The first item follows by simple algebraic manipulation. The second item is a fact from coding theory by notting that the largest code in $\mathbb{F}_2^m$ of distance 2 is smaller than the largest code in $\mathbb{F}_2^{m-1}$ of distance 1. □

B.2 Well-Formed Modulus & Ciphertext ZK Proof

Next we describe the missing ZK-proof $\Phi_{(\cdot)}$ [11] from Section 3.2. The proof is a combination of two Schnorr protocols $\Pi_{(\cdot)}$ and $\Theta_{(\cdot)}$ ("tight range proof" from [15] and "no small-factors proof" from [11]), and the ZK protocol Ξ ("Paillier-Blum proof" from [11]), all described below. Soundness and HVZK follow from the soundness and HVZK of the underlying protocols and Theorem 5.1. Recall that $\boldsymbol{I}(M) = \pm M/2$ and $\boldsymbol{J}(M) = \pm M \cdot 2^{\nu-1}$. Let $\pi = (\hat{N}, s, t, \ldots)$ for $s, t \in \mathbb{Z}_{\hat{N}}^*$. Recall that κ denotes the security parameter.

Paillier-Blum. The protocol is described in Figure 17. For the security properties (HVZK, soundness & Extraction) of Ξ, we refer the reader to [11, p. 28].

FIGURE 17 (Paillier-Blum Modulus ZK – $\Xi(N; p, q)$)

- **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 $\{y_i \leftarrow \mathbb{Z}_N^*\}_{i \in [\kappa]}$
3. For every $i \in [\kappa]$ set:
   - $x_i = \sqrt[4]{y_i'} \mod N$, where $y_i' = (-1)^{a_i} w^{b_i} y_i$ for unique $a_i, b_i \in \{0,1\}$ such that x_i is well defined.
   - $z_i = y_i^{N^{-1} \mod \phi(N)} \mod N$

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

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

Figure 17: Paillier-Blum Modulus ZK – $\Xi(N; p, q)$

Tight Range. Define Π_σ for $(\phi, \boldsymbol{E})$ for $\boldsymbol{E} = \boldsymbol{I}(2^\kappa)$, $\sigma = (\pi, T, \vec{A}, N)$ and

$$\begin{aligned}\phi : \mathbb{Z}^4 \times \mathbb{Z}^4 \times \mathbb{Z} \times \mathbb{Z}_N^* &\to \mathbb{Z}_{\hat{N}}^{*5} \times \mathbb{Z}_{N^2}^* \times \mathbb{G} \\ (x, \vec{\alpha}, \mu, \vec{\rho}, \beta, \gamma) &\mapsto (s^x t^\mu, (s^{\alpha_i} t^{\rho_i})_{i=1}^3, T^{-x} \cdot A_1^{\alpha_1} \cdot A_2^{\alpha_2} \cdot A_3^{\alpha_3} \cdot t^\beta, (1+N)^x \cdot \gamma^N, g^x)\end{aligned}$$

$(x, \vec{\alpha}, \mu, \vec{\rho}, \beta, \gamma) \in \boldsymbol{R}$ iff $x, \mu, \alpha_i, \rho_i \in \boldsymbol{I}(\hat{N} \cdot 2^\kappa)$, $\beta \in \boldsymbol{I}(\hat{N} \cdot 2^{2\kappa})$ and $\boldsymbol{S}$ is analogously defined with $\boldsymbol{J}(\cdot)$.

No Small Factors. Define $\Theta_{\pi,Q}$ for $(\phi, \boldsymbol{E}, \boldsymbol{R}, \boldsymbol{S})$ for $\boldsymbol{E} = \boldsymbol{I}(2^\kappa)$ and $\phi : \mathbb{Z}^2 \times \mathbb{Z}^3 :\to \mathbb{Z}_{\hat{N}}^{*3}$ s.t. $(p, q, u, v, w) \mapsto (s^p t^u, s^q t^v, Q^p t^w)$ with $(p, q, u, v, w) \in \boldsymbol{R}$ iff $p, q \in \boldsymbol{I}(\sqrt{N} \cdot 2^\kappa)$, $u, v \in \boldsymbol{I}(\hat{N} \cdot 2^\kappa)$ and $w \in \boldsymbol{I}(2^{2\kappa} \cdot \hat{N}\sqrt{N})$. Define $\boldsymbol{S}$ analogously with $\boldsymbol{J}(\cdot)$.

B.2.1 ZK Proof Description

Define Φ_π for $\pi = (\hat{N}, s, t, \dots)$ in Figure 18.

Prover

Common Input: N, W, X, κ

If not defined set $(W, X) = (1, \mathbb{1})$.

Secret Input: p, q, x, γ s.t. $N = pq$ and $W = \mathsf{enc}_N(x; \gamma)$, $X = g^x$ with $p, q \in \boldsymbol{I}(2^\kappa \sqrt{N})$ and $x \in [0, 2^\kappa]$.

1. Find $\alpha_1, \alpha_2, \alpha_3$ such that $4(2^\kappa - x) \cdot x + 1 = \alpha_1^2 + \alpha_2^2 + \alpha_3^2$
2. Sample $(\rho_i \leftarrow \boldsymbol{I}(2^\kappa \cdot \hat{N}))_{i=1,2,3}$ and set $\{A_i = s^{\alpha_i} t^{\rho_i} \mod \hat{N}\}_{i=1,2,3}$
3. Sample $\mu, u, v \leftarrow \boldsymbol{I}(2^\kappa \cdot \hat{N})$ and set $(S, T) = [(s^x t^\mu, (s^{2^\kappa} \cdot S^{-1})^4)]_{\hat{N}}$ and $(P, Q) = [(s^p t^u, s^q t^v)]_{\hat{N}}$
4. Set $\sigma = (\pi, T, \vec{A}, N)$ and generate Proofs:

$$\begin{cases} \psi \leftarrow \Pi_\sigma((S, \vec{A}, s, W, X); (x, \mu, \vec{\alpha}, \vec{\rho}, \beta, \gamma)) & \text{for } \beta = -4\mu x - \sum_{i=1}^{3} \alpha_i \rho_i \\ \xi \leftarrow \Theta_{\pi,Q}((P, Q, s^N); (p, q, u, v, w)) & \text{for } w = -pv \\ \eta \leftarrow \Xi((N); (p, q)) & \end{cases}$$

Output: $\zeta = (P, Q, S, \vec{A}, \psi, \xi, \eta)$.

Verifier

Common Input: N, W, X, κ.

Additional Input: Packing number λ & packing shift τ.

Proof: $\zeta \in \{0, 1\}^*$.

1. Parse $\zeta = (P, Q, S, \vec{A}, \psi, \xi, \eta)$.
2. Set $T = [(s^{2^\kappa} \cdot S^{-1})^4]_{\hat{N}}$ and $\sigma = (\pi, T, \vec{A}, N)$ and check that $N \geq 2^{\tau \cdot \lambda}$.
3. Verify $\psi(S, \vec{A}, s, W, X)$, $\xi(P, Q, s^N)$ and $\eta(N)$ according to Π_σ, $\Theta_{\pi,Q}$ and Ξ respectively.

Output: In case of failure output 0. Else, output 1.

Figure 18: Well-Formed Modulus & Ciphertext Φ_π

C Experimental Results

We present experimental results for evaluating the predominant cost of our protocol, i.e. computation complexity. So, in our experiments, we focus on *pure computation time during presigning and signing*; we view key generation as a one-time cost which does not affect performance in a significant way. Furthermore, the communication between the parties is managed in a simplified way (namely, all parties run on the same shared memory, and messages are "sent" and "received" by writing and reading the right memory slot). Finally, though we focus on computation, we recall that communication complexity is one of the main attractive features of our protocol, and, using our theoretical estimates (Table 1, p. 7), it is possible to infer real-world communication costs.

What we implement. We implement a proof of concept of the following phases of the protocol: key generation, presigning and signing. We recall that the signing phase consists of *init/CMP* and aggregation for the online parties, and ZK verification and output finalization for the offline party. Our proof of concept is written in C and the code is available online.[22] For elliptic curve operations, big numbers and hash functions, we use openSSL. We do not use other libraries.

What we measure. *Plot 1:* Computation time per message during presigning for the offline party as a function of the batching parameter, i.e. how many presignatures are handled in a single batch. *Plot 2:* Computation time per message during signing for each online party, as a function of the number of parties. *Plot 3:* Computation time for the aggregation phase during signing for each online party, as well as the verification and finalization time for the offline party, as a function of the batching parameter. *Plot 4:* Performance improvement (speedup) when using the packing optimization vs not using, as a function of the number of parties.

Choice of Parameters & Machine Specs. We instantiate the random oracle with SHA-512 (for Fiat-Shamir, commitments, etc...) and ECDSA is instantiated with hash function SHA-256 and elliptic curve secp256k1, i.e. the most popular variant of ECDSA in the blockchain space. The bit length of the Paillier modulus was accordingly set to 2048 to be eight times greater than the ECDSA key length. For the ZK proofs, we chose 64 bits for the (statistical) zero-knowledge parameter and 256 bits for the (computational) soundness parameter.

We performed our experiments on a MacBookPro with 2.4Ghz Quad-Core Intel Core i5 processor and 16 GB 2133 MHz LPDDR3 memory. Our experiments use single-threaded processes with default level of compilation optimization.

C.1 Concluding Remarks

Plot 1. We note that the batching technique significantly reduces the computation costs for presigning; e.g. for 9 parties, there is more than x2 speedup when batching 200 presignatures vs no batching. On the other hand, this speedup appears to plateau when batching more than a few hundred presignatures.

Plot 2. We observe a linear correlation between the number of parties and the computation time per message. This confirms our expectations, since each additional party adds a constant amount of work ($\approx$ 200ms).

Plot 3. Notice that the aggregation process for the online parties is tiny (at most 15ms for any number of parties) compared to the *init/CMP* part of the signing phase ($\approx$ 200ms with linear dependence on the number of parties, cf. Plot 3). We mention that there is a theoretical dependence on the number of parties even during aggregation; there is no such dependency for the offline party. However, the dependency is unnoticeable for small number of parties (e.g. fewer than 100). Finally, observe that the offline party's computational costs are rather insignificant during signing, and both of the aforementioned processes, i.e. aggregation for the online parties and verification/finalization for the offline, benefit from the batching technique.

[22] https://github.com/udi0peled/asymmetric_offline_cmp (accessed February 2023).

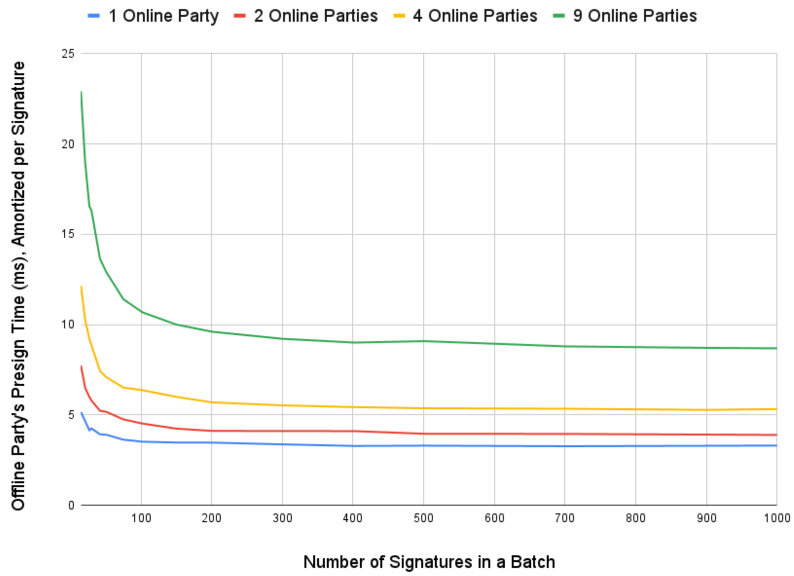

Figure 19: (Plot 1) CPU time for presigning for the offline party, viewed as a function of the batching parameter, i.e. number of future signatures in the batch. Reported values are amortized over the number of future signatures (so total costs scale linearly with respect to this quantity). We ran four different experiments for 1, 2, 4 and 9 online parties; recall that there is a single offline party and the protocol does not accommodate additional offline parties. (Computation time for each online party is small compared to the offline party and it does not increase with n – comparable to the blue line above)

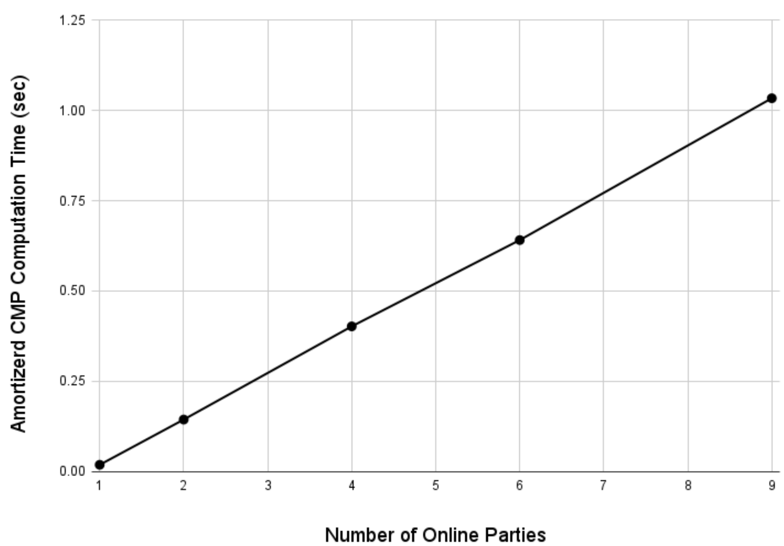

Figure 20: (Plot 2) Total CPU time for the online-signing phase, i.e. *init/CMP* + aggregation, for each online party (there is no offline party in this phase), per signature. The reported values were calculated by running the protocol 100 times and taking the average. We note that the bulk of the online-signing phase occurs during the *init/CMP* part of the protocol (cf. Plot 4 for the costs of the aggregation phase and those of the offline party).

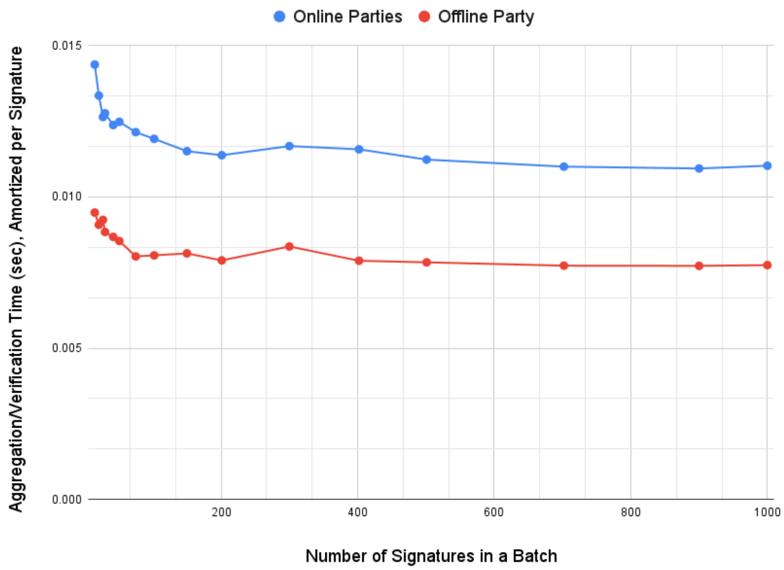

Figure 21: (Plot 3) CPU time for aggregation process for each online party and CPU running time for verification/finalization process for the offline party, viewed as a function of the batching parameter amortized over the number of signatures. We note that the displayed costs are not affected by the number of online parties (though there is a theoretical dependency for the blue line, cf. concluding remarks).

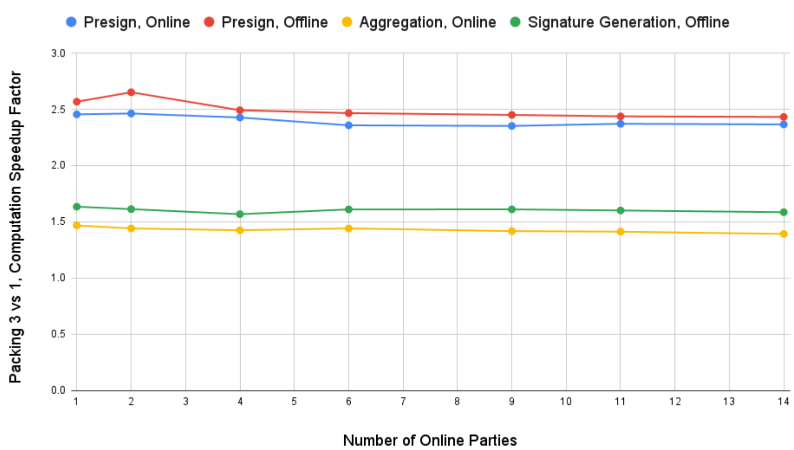

Figure 22: (Plot 4) CPU time speedup for presigning when packing three plaintexts into one Paillier ciphertext, compared to no packing at all, as a function of the number of online parties. E.g. when using packing number 3 during presigning, the parties run roughly 2.5 times faster compared to packing number 1. We do not report experiments for larger packing number (> 3) because the speedup deteriorates as the Paillier key length increases (cf. concluding remarks).

Plot 4. As mentioned in the caption of plot 4, the speedup deteriorates for larger packing because the Paillier plaintext size (and thus the key length) is increased to avoid overflow. As a consequence, the overhead of increasing the Paillier key length counteracts the benefits of packing (because Paillier encryption is basically the most expensive component of our protocol). However, it may yet be desirable to increase the packing number if the communication benefit outweighs the computation slowdown.

Comparison to the PCG-based protocol [1]. Form the offline party's perspective, or when viewed as a two-party protocol, we note that in [1] $\mathcal{P}_0$ receives 200B of data from the online word (compared to our 300B) and the authors estimate "1–2s per signature" [1, p. 27] (compared to our 100–200ms), so our protocol compares favorably to [1] in computation and the communication complexity of our protocol is within striking distance of [1] for the two-party case. For the multiparty case, [1] makes no distinction between online and offline signatories, so it exceeds the communication requirements of our use case, wastefully-so, because of the computational overhead.