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. Our protocol tolerates all-but-one *adaptive* corruptions, and it achieves full proactive security. Our protocol improves as follows over the state of the art.

- The preprocessing phase for calculating preprocessed data for future signatures is lightweight and non-interactive; it consists of each party 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 online "world" (approx. 300B).

We note that all previous ECDSA protocols require many rounds of interaction between all parties, and thus all previous protocols require extensive "interactive time" from the offline party. In contrast, our protocol requires minimal involvement from the offline party, and it is thus ideal for MPC-based cold storage.

Our main technical innovation for achieving the above is twofold: First, building on recent protocols, we design a two-party protocol that we non-generically compile into a highly efficient $(n+1)$-party protocol. Second, we present a new batching technique for proving in zero-knowledge that the plaintext values of practically any number of Paillier ciphertexts lie in a given range. The cost of the resulting (batched) proof is very close to the cost of the underlying single-instance proof of MacKenzie and Reiter (CRYPTO'01, IJIS'04).

We prove security in the UC framework, in the global random oracle model, assuming Strong RSA, semantic security of Paillier encryption, DDH, and enhanced existential unforgeability of ECDSA; these assumptions are widely used in the threshold-ECDSA literature and many commercially-available MPC-based wallets.

[*]Authors are listed in alphabetical order.

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

Contents

1 Introduction **3**
- 1.1 Related Work . . . 4
- 1.2 Our Results . . . 4
  - 1.2.1 Comparison . . . 5
  - 1.2.2 Security Features . . . 7
  - 1.2.3 Technical Overview . . . 7

2 Preliminaries **9**
- 2.1 Signatures and Unforgeability . . . 9
- 2.2 MPC and Universal Composability . . . 10
  - 2.2.1 Proactive Threshold Signatures . . . 11
  - 2.2.2 Ideal Threshold-Signature Functionality . . . 11
  - 2.2.3 Global Random Oracle . . . 12
- 2.3 Group/Number-Theoretic Definitions . . . 12
- 2.4 Schnorr Protocols . . . 12
  - 2.4.1 Embedded Schnorr Protocols . . . 13
- 2.5 NIZK and the Fiat-Shamir Transform . . . 13
  - 2.5.1 Proof Aggregation in ROM . . . 14

3 Protocol **15**
- 3.1 Baseline Two-Party Protocol . . . 15
  - 3.1.1 Key Generation and Refresh . . . 15
  - 3.1.2 Presigning . . . 17
  - 3.1.3 Signing . . . 17
- 3.2 Virtual Party MPC . . . 19
  - 3.2.1 `init` MPC Protocol (i.e. CMP) . . . 19

4 Security **20**
- 4.1 Unforgeability & Simulatability imply UC Security . . . 21
- 4.2 Simulatability of $\Sigma_{\mathtt{ecdsa}}$ . . . 21

5 Proof of Soundness for Batch-Proving **24**
- 5.1 Proof of Theorem 5.1 . . . 25
- 5.2 Putting Everything Together . . . 26
  - 5.2.1 Extractability . . . 26
  - 5.2.2 Robustness, Unpredictability and Binding . . . 28

References **28**

Ideal Threshold Signature Functionality **32**

Appendix **33**

A Missing ZK Protocols/Proofs **33**
- A.1 Security Proof for Multi-Pedersen Membership . . . 33
- A.2 Well-Formed Modulus & Ciphertext ZK Proof . . . 33
  - A.2.1 ZK Proof Description . . . 34

B Experimental Results **35**
- B.1 Concluding Remarks . . . 36

1 Introduction

The *digital signature algorithm* (DSA) [27] in its elliptic curve variant (ECDSA) [33] 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 naive solution; increase the number of signatories and require each transaction to be signed by different keys stored on separate machines (this can be achieved for example using Bitcoin's *multisig* functionality).

However, the resulting communication overhead may incur a substantial monetary cost, e.g. signatures must be stored on the blockchain and storing data on the blockchain is expensive. Thus, the naive solution is unattractive in many cases, especially when compared to threshold signatures, described next.

Threshold Signatures. Introduced by Desmedt [17] and Desmedt and Frankel [18], a t-out-of-n threshold signature scheme[1] 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, there are many companies that have implemented these protocols in their wallet infrastructure,[2] and the total transaction volume is estimated at *trillions* (of US dollars).[3]

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 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 some computation and communication capability (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.

Threshold ECDSA & Cold Storage. 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 is not connected to the internet) seems impractical given the current state of the art.[4]

[1]Threshold signatures are an instance of "Threshold Cryprography" which itself is one of the main application areas of the more general paradigm of Multi-Party Computation (MPC).

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

[3]Quote: "... surpassing \$2 trillion in assets transferred" retrieved from `fireblocks.com` (August 2022).

[4]One promising exception is the work of Abram et al. [1] which relies on *silent preprocessing* via *pseudorandom correlation generators* (PCGs). However, PCGs for ECDSA are based on fairly new cryptographic assumptions and real-world deployment is still at an early stage at the time of writing.

1.1 Related Work

Next, we discuss related work on threshold ECDSA [24]. We focus on trustless dishonest-majority protocols [31] and we refer the reader to the survey of Aumasson et al. [2] for a more thorough overview of general threshold-ECDSA protocols.

The first generation of trustless and (commercially) practical dishonest-majority protocols includes Castagnos et al. [11], Doerner et al. [19] and Lindell [28] for the two-party setting, and Castagnos et al. [12], Dalskov et al. [15], Doerner et al. [20], Gennaro and Goldfeder [22] and Lindell and Nof [30] for the multiparty setting. These protocols all achieve competitive performance with various tradeoffs between communication, computation and underlying cryptographic assumptions.

The second generation of dishonest-majority protocols includes Canetti et al. [9], Castagnos et al. [13], Gągol et al. [21] and Gennaro and Goldfeder [23]. These protocols typically include some added feature on top of dishonest security.[5] For instance, [9] achieves proactive security against adaptive adversaries (we discuss this security notion in Section 1.2.2), [23] achieves identifiable abort, i.e. the honest signatories can identify and expel any bad actors in case of a failed execution, and [21] achieves robustness after preprocessing, i.e. the protocol supports a preprocessing mode of operation that allows the signatories to execute (parts of) the protocol before the message to be signed is known, and the adversary cannot perform a DoS attack after the preprocessing phase is completed.

In fact, the preprocessing mode of operation (henceforth *pre-signing*), first observed in [15] and appearing independently in [9, 16, 21, 23], also offers a significant, often dramatic, performance improvement for most ECDSA protocols. Namely, in [9, 13, 15, 20, 23], pre-signing gives rise to a non-interactive signing phase which is *cheaper* than calculating a standard, non-threshold, ECDSA signature, because half the signature was precomputed during pre-signing.[6]

Why the above protocols fall short on cold storage? Unfortunately, the pre-signing mode of operation does not solve the MPC-cold-storage conundrum because while pre-signing renders the signing phase non interactive, pre-signing itself is a *highly* interactive protocol, so none of the previous protocols is particularly well suited for cold storage. In more detail, all of these protocols require at least three rounds of interaction in some stage (not including key generation) and each round requires heavy data processing that depends on the previous rounds. In terms of resources, per signature, all previous protocols either require *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).[7] In concrete terms, for 10000 signatures, or pre-signatures, the offline signatory has to send and receive gigabytes of data, or, it has to spend minutes to hours in computation time alone, all while running an interactive protocol with the online signatories.

1.2 Our Results

Building on the two-party protocols of Lindell [29], MacKenzie and Reiter [31] and the n-party protocol of Canetti et al. [9], 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 $\mathcal{P}_0$. Before we present the technical aspects of our protocol of our protocol, we first discuss its key characteristics and we compare to the state of the art. Our protocol boasts the following two new features: *Non-Interactive Pre-Signing* and *Asymmetric Signing*, both of which are lightweight. We expand on these features next.

Non-Interactive Pre-Signing. Similarly to other threshold-ECDSA schemes, our protocol supports a pre-signing mode of operation which allows the signatories to generate preprocessed data for *future* signing known as *pre-signatures*. However, unlike all previous protocols, the pre-signing phase of our protocol is completely non-interactive.[8] Namely, each signatory (whether online or offline) is instructed to send a single independently-generated Paillier ciphertext per pre-signature. We mention that *all* previous protocols require

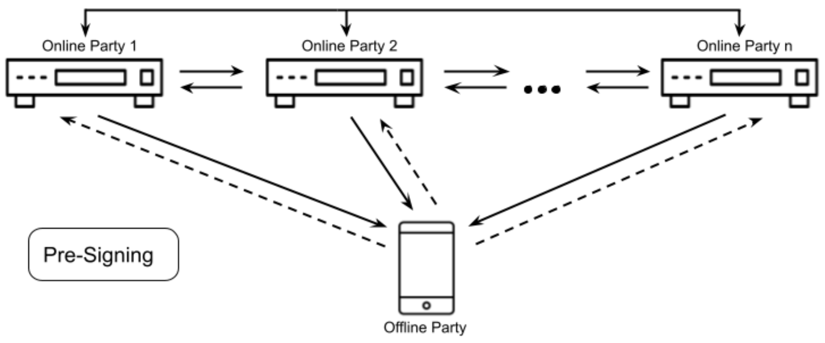
[5] [9] and [23] are independent works that were later combined into a single work [8, 10] covering (variations of) the two protocols.

[6] Non-threshold ECDSA may also enjoy this performance improvement, it goes without saying.

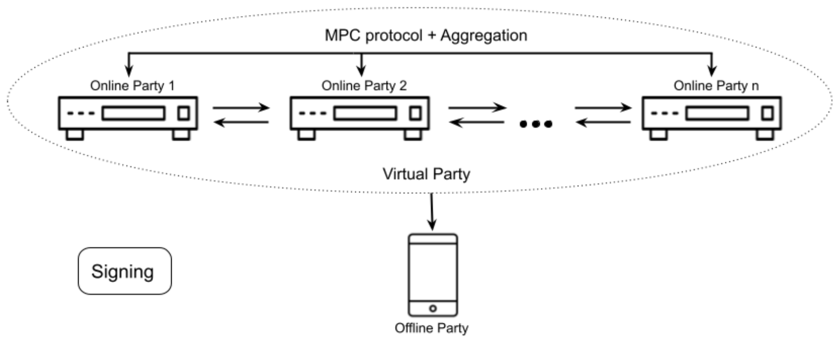
[7] Estimates are cited from [35], Table 2.

[8] For rushing adversaries (c.f. Section 2.2), the offline signatory is instructed to wait and receive the cosignatories' data before it sends its own.

many rounds of interaction, where each round involves heavy data processing that depends on the previous rounds.



Asymmetric Signing. Our protocol supports asymmetric signing, that is, from $\mathcal{P}_0$'s perspective, the signing phase to calculate the signature boils down to receiving a single message from the "online world". Furthermore, this message is short and it does not depend on the number of cosignatories. Specifically, $\mathcal{P}_0$ receives an encryption S of the desired signature under its own public key, and, to calculate the signature, it suffices to decrypt S and to verify an accompanying *short* ZK proof. As far as we know, our protocol is the first multiparty threshold-ECDSA protocol that simultaneously supports both non-interactive pre-signing and asymmetric signing under standard assumptions.[9]



The above make our protocol very attractive for MPC-based cold storage.

1.2.1 Comparison

In Table 1 we give a summary of the costs for the offline signatory of our protocol compared to the best performing two-party protocols under standard assumptions; further below we also compare to the PCG-based protocol [1] (which is based on a newer assumption, c.f. Footnote 4). Thus, the comparison from Table 1 may be interpreted in two ways; (1) Our protocol is the best-performing two-party ECDSA protocol, or (2) our protocol is the only multiparty protocol that can accommodate an offline signatory.

In concrete terms, for two cosignatories and 10000 presignatures, our protocol requires less than a minute of computation time and 3 megabytes of data, per party. Additionally, pre-signing is non-interactive and thus can be conveniently scheduled for downtime. To achieve such efficiency, we use a panoply of techniques that include *Party Virtualization and Proof Aggregation*, *Batch-Proving*, and *Packing* (c.f. Section 1.2.3).

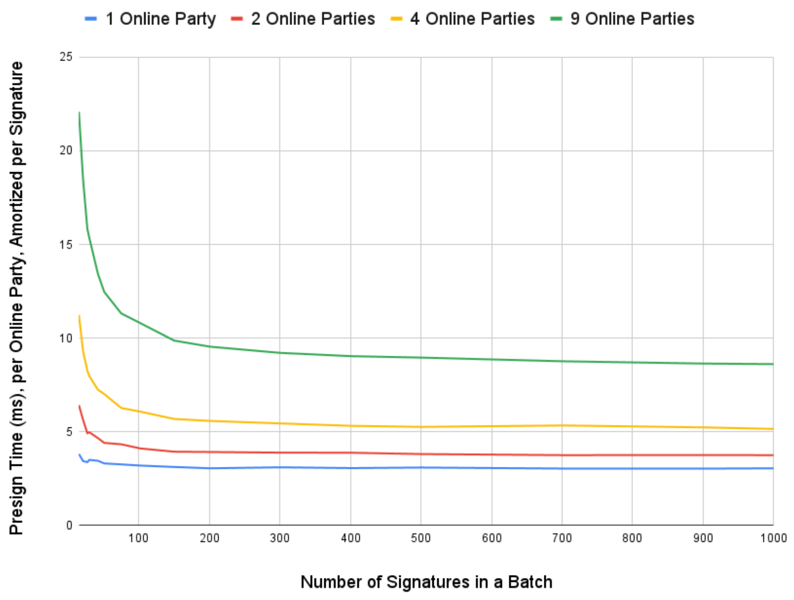
Costs for the Online Signatories. The computation and communication costs of our protocol are dominated by the complexity costs of the online signatories. However, these costs are very close to the costs of most state-of-the-art threshold-ECDSA protocols. In fact, there is a lot of flexibility on the choice of MPC

[9][9, 13, 15, 20, 23] trivially support asymmetric signing, but pre-signing is highly interactive. For two parties, [35] supports both asymmetric signing and non-interactive pre-signing (and [29, 31] can be suitably modified as well).

	Communication	Computation
This Work: Pre-Signing	$3n\boldsymbol{\mu}/p + 2n\boldsymbol{\gamma}$ ($n \cdot 300$B) in	$\boldsymbol{M}'/p + n(\boldsymbol{M}/p + 3\boldsymbol{m} + \boldsymbol{m}' + 2\boldsymbol{g})$
	$2\boldsymbol{\mu}/p + n\boldsymbol{\mu}/p + \boldsymbol{\gamma}$ ($n \cdot 300$B) out	$< \boldsymbol{M} + 2n \cdot (\boldsymbol{M} + \boldsymbol{g})$
This Work: Signing	$3\boldsymbol{\mu}/p + 3\boldsymbol{\gamma}$ (300B) in	$\boldsymbol{M}'/p + \boldsymbol{M} + \boldsymbol{m}' + 4\boldsymbol{g}$
	(0B) out	$< 3\boldsymbol{M} + 4\boldsymbol{g}$
Best Computation [35]	$8\boldsymbol{\gamma}^2$ (50KB)	$11\boldsymbol{g}$
Best Communication [35]	$16\boldsymbol{\mu} + 11\boldsymbol{\gamma}$ (6KB)	$14\boldsymbol{M} + 11\boldsymbol{g}$
Best Comm. (Class Groups) [11]	$14\boldsymbol{\gamma}$ (500B)	$4\boldsymbol{C} + 8\boldsymbol{g}$

Table 1: Costs for Offline Signatory vs Best 2PC ECDSA. In the above, 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. $\boldsymbol{M}$ and $\boldsymbol{M}'$ denote the unit cost of exponentiation modulo M and M^2, respectively, where M is a $\boldsymbol{\mu}$-bit RSA modulus ($\boldsymbol{m}$ and $\boldsymbol{m}'$ denote low-weight exponentiation, i.e. the bit length of the exponent is at least 10 times smaller than $\boldsymbol{\mu}$), and $\boldsymbol{C}$ denotes exponentiation in the relevant class group. When simplifying computational costs, we use the bounds $\boldsymbol{M}' \leq 3\boldsymbol{M}$, $\boldsymbol{m}' \leq 3\boldsymbol{m}$ and $\boldsymbol{m} \leq \boldsymbol{M}/2$. Similarly, $\boldsymbol{g}$ denotes the unit-cost of exponentiation in the ECDSA group (and $\boldsymbol{g}$ is much cheaper than either $\boldsymbol{M}$ and $\boldsymbol{C}$) and $\boldsymbol{\gamma}$ is the corresponding bit-length. Parameter $p \in \mathbb{N}$ represents the *packing number* (see Section 1.2.3) and it is assumed $p \geq 3$ in the concrete estimations. All costs are amortized over the number of signatures (or pre-signatures). For [11, 35], we report only one of either signing or pre-signing costs (the most expensive). We note that the comparison to multiparty protocols is even more favorable towards our protocol.

protocol that the online cosignatories execute, because the desired functionality is almost identical to the ECDSA functionality (c.f. Section 1.2.3). We instantiate the aforementioned functionality via the protocol from [9] (aka the CMP protocol – Figure 11) and our experimental results indicate 100–200ms in computation-time per signature per online party. (We refer the reader to Appendix B for a detailed picture of a number of experimental results). Below we display CPU running time (in ms) for presigning for each online party, viewed as a function of the batching parameter, i.e. number of future signatures in the batch. The plot shows 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. Notice that batching clearly improves (more than 2x in some cases) the performance of the pre-signing phase in computation time.



Comparison to the PCG-based protocol [1]. From the offline signatory's perspective, or when viewed as a two-party protocol, [1] requires 200B of data between each pair of parties (compared to our 300B) and the authors estimate "1–2s per signature" [1, p. 27] (compared to our 100–200ms). Furthermore, as a multiparty protocol, our protocol compares favorably to [1] in computation, and, since [1] makes no distinction between online and offline signatories, it exceeds the requirements of our use case in a wasteful way.

1.2.2 Security Features

Key-Refresh & Proactive Security. Our protocol supports a key-refresh phase where the signatories rerandomize all the secret material (ECDSA key-shares, Paillier keys, etc...) and it achieves full proactive security. To elaborate further, in the proactive paradigm, time is divided in epochs where the epochs are delineated by the key-refresh schedule. The adversary in the proactive paradigm is *adaptive*, i.e. it corrupts signatories dynamically throughout the evolution of the protocol, and it can adaptively choose which signatories to corrupt and/or decorrupt. Proactive security is defined as follows: a t-out-of-n threshold signing protocol achieves proactive security if the scheme remains unforgeable against any adversary that corrupts at most $t-1$ signatories in any given epoch. Note that most protocols are not even known to provide traditional (non-proactive) security against adaptive adversaries.[10]

Security & Composability. We use the proof technique from [9] to prove security in the Universal Composability (UC) framework, that is, we start with the ideal threshold signature functionality $\mathcal{F}_{\mathsf{tsig}}$ from [9] which captures all the desired security properties (unforgeability, proactive security, ...), and we show that our protocol UC-realizes $\mathcal{F}_{\mathsf{tsig}}$, i.e. the protocol emulates the ideal functionality, even when it is composed with other components of some larger system. One of our main technical contributions is a generalization of this technique to arbitrary threshold-signature schemes, discussed next.

1.2.3 Technical Overview

Unforgeability and Simulatability imply UC-Security. Our key technique (from [9]) is to show that our protocol UC-realizes $\mathcal{F}_{\mathrm{tsig}}$ by way of reduction to the assumed unforgeability of the underlying non-threshold scheme. In more detail, we describe a *straightline* simulator (in fact our simulator simply runs the honest parties' code when interacting with the environment), and the simulation is perfect, i.e. the real and ideal distributions are identically distributed, *unless* the environment can produce a forgery in the protocol, which, in turn, allows it to distinguish real from ideal execution. If so, there is a PPTM (the environment) that achieves some computational task (a forgery in our MPC protocol). Then, by way of reduction, we show that this PPTM can be used to achieve a different kind of computational task (a forgery for the underlying non-threshold signature scheme). Notice that this is just a run-of-the-mill reduction; therefore, the reduction may rewind the PPTM, and the reduction may provide simulated answers to the random oracle queries, i.e. the oracle is observable *and* programmable in the reduction. The former is the crux of the proof technique from [9] that remarkably enables UC-security against adaptive adversaries (which is notoriously difficult) without having recourse to sophisticated techniques like non-committing encryption.

In this paper, we generalize the above so that it is applicable to general threshold protocols. So, starting with an arbitrary, non-threshold, signature scheme and the standard notion of unforgeability,[11] we define threshold-signature *protocols* and the notion of *simulatability* which is the analogue of unforgeability for protocols, i.e. one can simulate the adversary's view using the signing oracle from the unforgeability game. Our main technical lemma (Theorem 4.3) states that if the protocol is simulatable against adaptive adversaries, and the underlying signature is unforgeable, then the protocol UC-realizes functionality $\mathcal{F}_{\mathrm{tsig}}$ against adaptive adversaries.

Party Virtualization & Proof Aggregation. Let $(\mathbb{G}, g, q)$ denote the group-generator-order tuple for ECDSA. For this high-level overview, we will abuse notation and write $R \cdot x \in \mathbb{F}_q$ to denote the field element

[10] We mention that some protocols achieve the weaker notion of proactive security against static adversaries i.e. the corruption pattern for all (future) epochs is determined at the beginning of the execution.

[11] To support added functionality, e.g. pre-signing, and to achieve the utmost generality, we augment the traditional notion of unforgeability in the technical sections.

obtained by projecting[12] $R \in \mathbb{G}$ 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 $R = g^{k^{-1}} \in \mathbb{G}$. Effectively, all multiparty ECDSA protocols realize the following functionality

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

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. The resulting ECDSA signature is obtained by publishing and summing up $\{\sigma_i\}_{i \in [n]}$ for $\sigma_i = k_i m + R \cdot \chi_i$.

Turning to our $(n+1)$-party protocol, write $\mathcal{P}_0$ for the offline signatory and $\mathcal{P}_1, \ldots, \mathcal{P}_n$ for the online signatories, and let $x_0, x_1, \ldots, x_n$ denote an additive sharing of the secret key. Our breakthrough observation is that using an encryption of x_0 under an additive-homomorphic scheme, and running $\mathtt{tecdsa}_{(\cdot)}$ on a point H provided by $\mathcal{P}_0$, the online signatories $\mathcal{P}_1, \ldots, \mathcal{P}_n$ can calculate an encryption of a "partial" signature which $\mathcal{P}_0$ can complete into a "full" signature. In more detail:

0. Offline signatory $\mathcal{P}_0$ sends $W_0 = \mathsf{enc}_0(x_0)$ and $H = g^{\alpha^{-1}} \in \mathbb{G}$ to the online signatories $\mathcal{P}_1, \ldots, \mathcal{P}_n$.

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

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. When obtaining the hash m of the desired message, each online 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 and the (aggregated) ciphertext $S = \oplus_{i=1}^n S_i$ to $\mathcal{P}_0$. .

3. Offline signatory calculates $\hat{\sigma} = \mathsf{dec}_0(S)$ an outputs the ECDSA signature (R, σ) for $\sigma = \alpha \cdot \hat{\sigma}$.

Item 2 above is equivalent to $\mathcal{P}_0$ receiving a message from a single *virtual* party and this message is independent of the number of parties. Furthermore, for malicious security, S is accompanied with a ZK proof ψ that it is well formed and the proof ψ itself is the result of some aggregation process among the online signatories. From a technical standpoint, aggregation is feasible (and lightweight) thanks to the homomorphic properties of the underlying encryption scheme and proof system, explained next.

Schnorr Protocols & Batch-Proving. 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 [32], we define Schnorr protocols abstractly (c.f. Section 2.4) as protocols for proving that the preimage $w \in \mathbb{H}$ of a group element $X \in \mathbb{G}$ by some homomorphism $\phi : \mathbb{H} \to \mathbb{G}$ lies in a given subset $\boldsymbol{R} \subseteq \mathbb{H}$. Then, we prove a general theorem for batching many instances $X_1, \ldots, X_\ell$ into a single proof ψ (obtained via the Fiat-Shamir transform, c.f. Section 2.5.1). While Schnorr protocols are known to admit batching capabilities, e.g. Gennaro et al. [25] and Thyagarajan et al. [34], previous results do not account for our use case,[13] and the unified view that we propose is new. In Table 2, we provide a comparison of our flavor of batch-proving vs the baseline protocol of MacKenzie and Reiter [31], and the batch-proving protocol of Thyagarajan et al. [34]; the communication complexity of our scheme compares favorably by orders of magnitude for any batch size (we note that [34] overtakes our protocol in computation for large batches).

Packing. In general, *packing* refers to the principle of storing many (small) values in a single (large) unit of space, e.g. encrypt w_1, w_2 as $C = \mathsf{enc}(w_1 + 2^\tau w_2)$ for suitable τ assuming the bit-length of the plaintext-space is larger than $|w_1| + |w_2|$. We show that our protocol enjoys a performance improvement thanks to this folklore technique and total communication is reduced by a multiplicative factor p (e.g. $p \geq 3$ for secp256k1 for suitable choice of parameters).[14]

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

[13]As far as we can tell, previous results relied on the fact that ϕ is injective, which is not the case for us, because, getting ahead of ourselves, Pedersen commitments are only computationally binding.

[14]While the general principle is folklore, there were many technical challenges for showing that our protocol supports packing, e.g. Pedersen Proof of knowledge is packable (c.f. Theorem 5.1) or Multi-Pedersen commitments are binding (c.f. Fact 5.10).

	Comm.	Prover Comp.	Verifier Comp.
MacKenzie and Reiter [31]	$\ell \cdot 5\boldsymbol{\mu}$	$\ell \cdot (\boldsymbol{M}' + \boldsymbol{M} + 2\boldsymbol{m})$	$\ell \cdot (\boldsymbol{M}' + \boldsymbol{M} + \boldsymbol{m}' + 2\boldsymbol{m})$
		$\approx \ell \cdot (4\boldsymbol{M} + 2\boldsymbol{m})$	$\approx \ell \cdot (4\boldsymbol{M} + 5\boldsymbol{m})$
Thyagarajan et al. [34]	$\kappa \cdot 3\boldsymbol{\mu}$	$\kappa \cdot \boldsymbol{M}'$	$\kappa \cdot \boldsymbol{M}'$
		$\approx \kappa \cdot 3\boldsymbol{M}$	$\approx \kappa \cdot 3\boldsymbol{M}$
Batch-Proving (**Our Work**)	$5\boldsymbol{\mu}$	$\boldsymbol{M}' + \boldsymbol{M} + (\ell + 1) \cdot \boldsymbol{m}$	$\boldsymbol{M}' + \boldsymbol{M} + \boldsymbol{m} + \ell \cdot (\boldsymbol{m}' + \boldsymbol{m})$
		$\approx 4\boldsymbol{M} + (\ell + 1) \cdot \boldsymbol{m}$	$\approx 4\boldsymbol{M} + \boldsymbol{m} + \ell \cdot 4\boldsymbol{m}$
		$\leq 4\boldsymbol{M} + \ell \cdot \boldsymbol{M}/2$	$\leq 4\boldsymbol{M} + \ell \cdot 2\boldsymbol{M}$

Table 2: In the ROM, for security parameter κ, comparison of [31, 34] with our work for proving that the plaintext values of ℓ Paillier ciphertext lie in a given range. The values $\boldsymbol{\mu}, \boldsymbol{m}, \boldsymbol{m}', \boldsymbol{M}, \boldsymbol{M}'$ are as in Table 1, and the target range is $[-2^{\boldsymbol{\mu}/10}, 2^{\boldsymbol{\mu}/10}]$. The (baseline) protocol of [34] achieves constant soundness and requires amplification via parallel repetition; hence the dependency on κ.

2 Preliminaries

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

Notation. Throughout the paper $\mathbb{G}, \mathbb{H}, \mathbb{K}$ denote groups and $\mathbb{F}$ is a field (typically we write $\mathbb{F}_q$ to specify that the fields 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. We let $\mathbb{Q}, \mathbb{Z}, \mathbb{N}$ denote the set of rational, integer and natural numbers, respectively. We use sans-serif letters ($\mathsf{enc}, \mathsf{dec}, \ldots$) or calligraphic ($\mathcal{S}, \mathcal{A}, \ldots$) to denote algorithms. Secret values are always denoted with lower case letters ($p, q, \ldots$) and public values are *usually* denoted with upper case letters ($A, B, N, \ldots$). Furthermore, for a tuple of both public and secret values, e.g. an RSA modulus and its factors (N, p, q), we use a semi-colon to differentiate public from secret values (so we write $(N; p, q)$ instead of (N, p, q)). 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 $\boldsymbol{X}$. Bold letters may also denote random variables. Arrow-accented letters $\vec{A}, \vec{\rho}, \ldots$ denote ordered sets, i.e. tuples. 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 $\{-|\ell|, \ldots, 0, \ldots, |\ell|\}$. We write $x \leftarrow \boldsymbol{E}$ for sampling x uniformly from a set $\boldsymbol{E}$, and $x \leftarrow \mathcal{A}$ or $x \leftarrow \mathsf{gen}$ for sampling x according to (probabilistic) algorithms $\mathcal{A}$ or gen. 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 indistingushable and we write $\{\boldsymbol{v}_\kappa\} \equiv \{\boldsymbol{u}_\kappa\}$ if $\Pr[\mathcal{D}(1^\kappa, \boldsymbol{u}_\kappa) = 1] - \Pr[\mathcal{D}(1^\kappa, \boldsymbol{v}_\kappa) = 1]$ is negligible for every efficient distinguisher $\mathcal{D}$. We write $\mathrm{SD}(\boldsymbol{u}, \boldsymbol{v})$ for the statistical distance of $\boldsymbol{u}$ and $\boldsymbol{v}$. For $a, b \in \mathbb{N}$, we write $a \mid b$ for "a divides b" and $a \not| \, b$ for the negation. Finally, $\gcd : \mathbb{N}^2 \to \mathbb{N}$ denotes the gcd 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 usually denotes a group homomorphism).

2.1 Signatures and Unforgeability

Definition 2.1 (Signature Scheme.)**.** $\mathrm{Sig} = (\mathsf{gen}, \mathsf{sign}, \mathsf{vrfy})$ 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 $\mathsf{vrfy}_{\mathsf{pk}}(\sigma, \mathrm{msg}) = 1$.

FIGURE 1 (Augmented signature oracle $\mathcal{G}$)

Parameters. Signature scheme Sig and randomized functionality $\mathcal{F}$.

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 $(\mathcal{F}, x)$, sample $r \leftarrow \$$ and return $\tau = \mathcal{F}(x, \mathtt{state}; r)$.
   Update $\mathtt{state} := \mathtt{state} \cup \{(x, \tau; r)\}$.

Figure 1: Augmented signature oracle $\mathcal{G}$

FIGURE 2 ($\mathcal{G}$-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 $\mathcal{G}$.
3. $\mathcal{A}$ outputs (m, σ) given its view (randomness and query-answer pairs to $\mathcal{G}$)

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

Figure 2: $\mathcal{G}$-Existential Unforgeability Experiment $\mathcal{G}\text{-}\mathsf{EU}(\mathcal{A}, 1^\kappa)$

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

Definition 2.2 ($\mathcal{G}$-Existential Unforgeability.)**.** We say that Sig satisfies $\mathcal{G}$-Existential unforgeability if there exists $\nu \in \mathsf{negl}(\kappa)$ such that for all $\mathcal{A}$, it holds that $\Pr[\mathcal{G}\text{-}\mathsf{EU}(\mathcal{A}, 1^\kappa) = 1] \le \nu(\kappa)$, where $\mathcal{G}\text{-}\mathsf{EU}(\cdot)$ denotes the security game from Figure 2.

2.2 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.

In this work 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 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 [5, 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 $\mathcal{Z}$ it holds that

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

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 $\mathcal{G}$.

2.2.1 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 $\mathbb{Q} \subseteq 2^{\boldsymbol{P}}$. We say that Σ is a proactive threshold-signature scheme for $\mathrm{Sig} = (\ldots, \mathsf{vrfy})$ 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 $\mathcal{P}_i \in \boldsymbol{Q} \in \mathbb{Q}$ 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 $\mathcal{P}_i \in \boldsymbol{Q} \in \mathbb{Q}$ and returns σ to (at least one) $\mathcal{P}_i$.

   Correctness. Using the notation above, it holds that $\mathsf{vrfy}_{\mathsf{pk}}(\sigma, \mathrm{msg}) = 1$ in an honest execution.

Sets $\boldsymbol{Q} \in \mathbb{Q}$ 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 $\mathcal{F}_{\mathrm{tsig}}$, defined below.

2.2.2 Ideal Threshold-Signature Functionality

We use the ideal functionality $\mathcal{F}_{\mathsf{tsig}}$ of [9], which generalizes the non-threshold signature functionality of Canetti [6]. We briefly outline $\mathcal{F}_{\mathsf{tsig}}$ next and we refer the reader to the appendix (p. 32) 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 functional record 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.

2.2.3 Global Random Oracle

We follow formalism of [4, 7] 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 3 (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 3: The Global Random Oracle Functionality $\mathcal{H}$

2.3 Group/Number-Theoretic Definitions

Definition 2.5. We say that $N \in \mathbb{N}$ is a biprime if N admits exactly two non-trivial divisor. 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 $r = (g^m \cdot X^r)^{\sigma^{-1}}|_{x\text{-axis}} \mod q$.

Notation 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 $\beta \leftarrow \mathbb{F}_q$.

2.4 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 the security parameter.

Definition 2.9. An *m-batched* Schnorr protocol Π for tuple $(\phi, \boldsymbol{E}, \boldsymbol{R}, \boldsymbol{S})$ consists of the following protocol. 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 $\tau = (A, \vec{e}, z)$ 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 the probability that the verifier rejects with probability at least ν.

Further define

$(\varepsilon, \boldsymbol{V})$-Extractibility. For all efficient $\mathcal{A}$, if $\{\tau_j = (A, \vec{e}_j, z_j)\}_{j=1}^{m+1} \leftarrow \mathcal{A}$ 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 $\alpha, \{w_j\}_j \leftarrow \mathcal{E}(\{X_j, \tau_j\}_{j=1}^{m+1})$.

2.4.1 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-batched *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 protocol. 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.

HVZK, *soundness* and *extractability* are defined analogously to non-embedded protocols.

2.5 NIZK and the Fiat-Shamir Transform

Fiat-Shamir. 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 4. (The process is analogous for embedded Schnorr protocols)[15]

Definition 2.11. We say that $\psi = (A, \vec{e}, z)$ is a valid *proof* for (X, aux) 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 is a secure proof system in the ROM if

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

FIGURE 4 (Schnorr Proof in ROM $\psi \leftarrow \Pi(\mathsf{aux}, \vec{X}; \vec{w})$)

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

Zero-Knowledge. If $X_i = \phi(w_i)$ and $w_i \in \boldsymbol{R}$ for all i, then $\tau = (A, \vec{e}, z)$ 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 $\phi(w_j) \neq X_j$, for every $w \in \boldsymbol{S}$, then the probability that an efficient PPTM outputs a valid proof is negligible.

Fact 2.12. *If Π is (μ, ν)-secure for $\mu, \nu \in \mathsf{negl}(\kappa)$, then Π is a secure proof system in the ROM.*

2.5.1 Proof Aggregation in ROM

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

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

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

$$\xrightarrow{C_i} \quad \xleftarrow[C_j]{}$$

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

$$\xrightarrow{A_i, \rho_i} \quad \xleftarrow[A_j, \rho_j]{}$$

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

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

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

$$\xrightarrow{z_i} \quad \xleftarrow[z_j]{}$$

Output $\psi = (A, \vec{e}, z)$ if

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

Figure 5: NIZK Aggregation for common input $\vec{X}$

Let Π denote (μ, ν)-secure Schnorr protocol for $(\phi, \boldsymbol{E}, \boldsymbol{R}, \boldsymbol{S})$. For $\mathcal{A}$ controlling a strict subset of the provers in Figure 5, write $\mathrm{Real}_\mathcal{A}$ for the adversary's view in an execution of the protocol. For simulator $\mathcal{S}$ taking auxiliary input $\{X_{j,\ell} = \phi(w_{j,\ell})\}_{j \notin \mathcal{A}, \ell}$ with blackbox access to $\mathcal{A}$ and oracle access to $\mathcal{H}$, write $\mathrm{Ideal}_\mathcal{S}$ for the output of $\mathcal{S}$.

Claim 2.13. *For every $\mathcal{A}$, $\vec{X} \in \mathbb{G}^m$, $\mathsf{aux} \in \{0,1\}^*$, exists $\mathcal{S}$ in the (observable & programmable) ROM s.t.*

$$\mathrm{SD}(\mathrm{Real}_\mathcal{A}, \mathrm{Ideal}_\mathcal{S}) \leq n \cdot \mu.$$

3 Protocol

In this section we define our proactive threshold-ECDSA protocol $\Sigma_{\mathsf{ecdsa}} = (\Sigma_{\mathsf{kgen}}, \Sigma_{\mathsf{refr}}, \Sigma_{\mathsf{pres}}, \Sigma_{\mathsf{sign}})$. We begin by presenting the protocol in its two-party variant (Section 3.1). For the party-virtualization step (Section 3.2), we give full details only for the signing phase Section 3.1.3, since the other phases of the protocol can be virtualized easily (e.g. presigning) or they are very close to protocols in the literature (e.g. key generation & key refresh from [9]). During signing, it is assumed 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.

Parameters. Hereafter, ℓ, ε s.t. $\ell, \varepsilon - \ell \in O(\kappa)$ and $\ell \approx \log(q)$. For $M \in \mathbb{N}$, let $\boldsymbol{I}(M) = \pm M$ and $\boldsymbol{J}(M) = \pm M \cdot 2^{\varepsilon}$.

Definition 3.1 (Pedersen Parameters)**.** Define algorithm ped that takes input 1^κ and outputs $\pi = (\hat{N}, t, s_1, \ldots, s_m, \psi)$ such that

1. $\hat{N} = pq$ of size $\lambda \in O(\kappa)$ such that $p = 2p' + 1$ and $q = 2q' + 1$ and p', q' are prime as well.
2. $t \in \mathrm{QR}(\mathbb{Z}^*_{\hat{N}})$ and $s_1, \ldots, s_m \in \langle t \rangle$.
3. $\psi \leftarrow \Pi(s_1, \ldots, s_m)$ where Π is the batched prot. for $(\phi, \boldsymbol{E})$ where $\boldsymbol{E} = \{0, 1\}$ and $\phi(\alpha) = t^\alpha \mod \hat{N}$.

3.1 Baseline Two-Party Protocol

Each phase of the protocol is defined in Figure 6 (key generation Σ_{kgen}), Figure 7 (key refresh Σ_{refr}), Figure 8 (presigning Σ_{pres}) and Figure 9 (signing Σ_{sign}). To help the reader understand the multiparty case later, we have opted to describe the online party $\mathcal{P}_\infty$ *as if* it locally emulates (mocks) the cosignatories $\mathcal{P}_1, \ldots, \mathcal{P}_n$.

Notation and Conventions. Let λ and m denote respectively the packing and batching number. Let τ is the pack shift, i.e. $w_0 + 2^\tau w_1$ is a packing of w_0 and w_1. The parties 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 $\psi \leftarrow \Pi(\mathsf{aux}, X; w)$ (we simply write $\Pi(X; w)$ to reduce clutter) where Π is a Schnorr protocol, and each protocol is described at the beginning of each subsection, when needed. It is stressed that most Schnorr protocol herein are *embedded* Schnorr protocols (c.f. Definition 2.10).

To avoid clutter, and to improve readability, we have 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 $\mathcal{P}_i$ where the randomizer is chosen as prescribed. Note that the randomizers are important for the 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, the parties are instructed to verify a given proof against the available data, every time one is received.

3.1.1 Key Generation and Refresh

The key generation (c.f Figure 6) contains a ZK protocol, Φ_π, which cannot be cast entirely as Schnorr protocol. We note that it yields the validity of the tuple (N, W, X, q) for the following NP relation, and we defer the details to Appendix A.2.[16] Namely, for $(N, W, X, q) \in \mathbb{N} \times \mathbb{Z}^*_{N^2} \times \mathbb{G} \times \mathbb{N}$, integer N is a Paillier-Blum modulus and N does not admit small factors (smaller than q). Furthermore $W \in \mathbb{Z}^*_{N^2}$ is an encryption of $x \in [0, q-1]$ under Paillier key N, and $X = g^x \in \mathbb{G}$. If W and X are not specified, it is assumed that $(N, W, X, q) = (N, 1, \mathbb{1}, q)$.

“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 $V_0 = \mathcal{H}(\mathsf{aux}, \mathcal{P}_0, X_0, A_0, N_0, W_0, Y_0, \pi_0, u_0)$
for $(X_i, A_i) = (g^{x_0}, g^{\alpha_i})$		for $(X_0, A_0) = (g^{x_i}, g^{\alpha_i})$, $W_0 = \mathsf{enc}_0(x_0)$
	$\xrightarrow{V_1, \ldots, V_n}$ $\xleftarrow[V_0]{}$	
	$\xrightarrow{(X_i, A_i, Y_i, N_i, \pi_i, u_i)_i}$ $\xleftarrow[X_0, A_0, Y_0, N_0, W_0, \pi_0, u_0]{}$	
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
$\forall i$, $\psi_i \leftarrow \Phi_{\pi_0}(N_i; p_i, q_i)$		$\forall i$, $\psi'_i \leftarrow \Phi_{\pi_i}(N_0, W_0, X_0; p_0, q_0, x_0)$
$e_i = \mathcal{H}(\mathcal{P}_i, X_i, A_i, u)$ 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 6: Threshold ECDSA: Key Generation (Σ_{kgen})

“Online” Party $\mathcal{P}_\infty$		“Offline” Party $\mathcal{P}_0$
Execute Key-Generation (Figure 6)		Execute Key-Generation (Figure 6)
with key shares $(x_i)_{i=1}^n$		with key share $\hat{x}_0 = [x_0 + \alpha]_q$ for $\alpha \leftarrow \mathbb{F}_q$
Obtain $(\hat{Y}, \hat{X}_0, \hat{N}_0, \hat{W}_0, (X_i, \hat{N}_i, \hat{\pi}_i)_i; (x_i)_i)$		Obtain $(\hat{Y}, \hat{X}_0, \hat{N}_0, \hat{W}_0, \hat{\pi}_0, X_\infty, (\hat{N}_i, \hat{\pi}_i)_i); \hat{x}_0)$
	$\xleftarrow{\alpha}$	
Verify $X_0 \cdot g^\alpha = \hat{X}_0$		
Sample $\beta_1, \ldots, \beta_n \leftarrow \mathbb{F}_q$ s.t. $\alpha = \sum_i \beta_i$		
Set $(\hat{x}_i, \hat{X}_i)_{i=1}^n := (x_i + \beta_i, X_i \cdot g^{\beta_i})_{i=1}^n$		Set $\hat{X}_\infty := X_\infty \cdot g^{-\alpha}$
Output $(\hat{Y}, \hat{X}_0, \hat{N}_0, \hat{W}_0, (\hat{X}_i, \hat{N}_i, \hat{\pi}_i)_i; (\hat{x}_i)_i)$		Output $(\hat{Y}, \hat{X}_0, \hat{N}_0, \hat{W}_0, \hat{\pi}_0, \hat{X}_\infty, (\hat{N}_i, \hat{\pi}_i)_i); \hat{x}_0)$

Figure 7: Threshold ECDSA: Refresh (Σ_{refr})

"Online" Party $\mathcal{P}_\infty$		"Offline" Party $\mathcal{P}_0$
For $i \in [n]$, $j \in [m]$, $t \in [\lambda]$, do:		
$k_{i,j,t} \leftarrow \mathbb{F}_q,\ \vec{B}_{i,j,t} = \mathsf{com}(k_{i,j,t}, Y)$		
$K_{i,j} = \mathsf{enc}_i(\sum_{t=1}^{\lambda} 2^{\tau(t-1)} \cdot k_{i,j,t})$		
$\forall i,\ \psi_i \leftarrow \Xi_0((K_{i,j}, \vec{B}_{i,j,t})_{j,t}; (k_{i,j,t})_{j,t})$		
	$\xrightarrow{(\psi_i, (K_{i,j}, (\vec{B}_{i,j,t})_t)_j)_i}$	
		For $j \in [m]$, $t \in [\lambda]$, do:
		$\alpha_{j,t} \leftarrow \mathbb{F}_q$, ; $H_{j,t} = g^{\alpha_{j,t}^{-1}}$
		$C_j = \mathsf{enc}_0(\sum_{w=0}^{\lambda-1} 2^{\tau(t-1)} \cdot \alpha_{j,t}^{-1})$
		$\forall i, \psi'_i \leftarrow \Xi_i((C_j, (H_{j,t})_t)_j; (\alpha_{j,t})_{j,t})$
	$\xleftarrow{(\psi'_i)_i, (C_j, (H_{j,t})_j)_t}$	
$\forall i$ output		$\forall j, t$ set $\vec{B}_{j,t} = \prod_{i=1}^{n} \vec{B}_{i,j,t}$
$(k_{i,j,t}, \vec{B}_{1,j,t}, \ldots, \vec{B}_{n,j,t}, H_{j,t})_{j,t}$		output $(\alpha_{j,t}, H_{j,t}, \vec{B}_{j,t})_{j,t}$

Figure 8: Threshold ECDSA: Presigning (Σ_{pres})

3.1.2 Presigning

Hereafter, we assume that all the signing material (Paillier/El-Gamal Keys, Pedersen parameters,...) are fixed according to Σ_{kgen} (or Σ_{refr}), e.g., unambiguously, Y refers to the El-Gamal key and (N_i, π_i) refers to the Paillier key and Pedersen parameters of party $\mathcal{P}_i$, as chosen in Σ_{kgen} (or Σ_{refr}).

Definition 3.2. 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 $\boldsymbol{E} = \boldsymbol{I}(2^\ell)$

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

and

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

and

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

3.1.3 Signing

Without loss of generality, we assume that the signing phase consumes all the available presignatures.

Definition 3.3. Define Θ_R to be the Schnorr protocol associated with $(\phi, \boldsymbol{E})$ for $\boldsymbol{E} = \boldsymbol{I}(2^t)$ and

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

[16] Φ_π is a straightforward combination of proofs found in [9] and [14].

"Online" Party $\mathcal{P}_\infty$ "Offline" Party $\mathcal{P}_0$

For $j \in [m]$, $t \in [\lambda]$, retrieve $\vec{B}_{j,t}$ and do:

Initialize:

Set $r_{j,t} = H_{j,t}^{k_{j,t}^{-1}}|_{\text{x-axis}}$ for $k_{j,t} = \sum_i k_{i,j,t}$

Set $\chi_{j,t} = [x_\infty \cdot k_{j,t}]_q$ and $\vec{U}_{j,t} = \mathsf{com}(\chi_{j,t}, Y)$

Prepare Payload:

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

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

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

Generate Proofs:

$\psi_{j,t} \leftarrow \Theta_{R_{j,t}}((\vec{B}_{j,t}, H_{j,t})_t; (k_{j,t})_j)$

$\psi' \leftarrow \Theta'(((\vec{V}_{j,t}, \vec{Z}_{j,t})_t, S_j)_j; (\zeta_{j,t}, \mu_{j,t})_j)$

for $\vec{V}_{j,t} = \vec{B}_{j,t}^{r_{j,t}}$ and $\vec{Z}_{j,t} = \vec{B}_{j,t}^{m} \cdot \vec{U}_{j,t}^{r_{j,t}}$

$\psi'' \leftarrow \Theta''((\vec{B}_{j,t}, \vec{U}_{j,t})_{j,t}; (k_{j,t})_{j,t})$

$$\xrightarrow{\psi', \psi'', ((\vec{U}_{j,t}, r_{j,t}, \psi_{j,t})_t, S_j)_j}$$

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

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

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

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

Figure 9: Threshold ECDSA: Signing (Σ_{sign})

Definition 3.4. For $(\hat{N}, t, s_1, r_1, \ldots) = \pi_0$, define Schnorr protocol Θ' for $(\phi, \boldsymbol{E}, \boldsymbol{R}, \boldsymbol{S})$ where $\boldsymbol{E} = \boldsymbol{I}(2^\ell)$

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

and

$$\begin{cases} \phi_1 : (\vec{w}, \vec{\mu}) \mapsto (g^{\mu_j}, Y^{\mu_j} \cdot g^{w_j})_{j=1}^{\lambda} \\ \phi_2 : (\vec{w}, \vec{z}, \nu) \mapsto \prod_{j=1}^{\lambda} (1 + 2^{\tau \cdot (j-1)} \cdot N_0)^{w_j} \cdot W_0^{z_j} \cdot \nu_0^N \mod N_0^2 \\ \phi_3 : (\vec{w}, \rho) \mapsto t^\rho \cdot \prod_{j=1}^{\lambda} s_j^{w_j} r_j^{z_j} \mod \hat{N} \end{cases}$$

and

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

In words, the embedded Schnorr proof Θ' yields that for $((\vec{U}_j, \vec{V}_j)_j, C) \in \mathbb{G}^{4\lambda} \times \mathbb{Z}_{N^2}^*$, the packed ciphertext C was calculated as a weighted combination above using the secrets fro $\{\vec{U}_j, \vec{V}_j\}$

Definition 3.5. Define Θ'' to be the Schnorr protocol associated with tuple $(\phi, \boldsymbol{E})$ for $\boldsymbol{E} = \boldsymbol{I}(2^\ell)$, $\phi : \mathbb{F}_q^3 \to \mathbb{G}^4$ s.t. $(k, b, v) \mapsto (g^b, Y^b \cdot g^k, g^v, Y^v \cdot X_\infty^k)$.

In words, Θ'' yields that for $(\vec{B}, \vec{U}) \in \mathbb{G}^2 \times \mathbb{G}^2$, $\vec{B}$ and $\vec{U}$ hide the same secret base g and X_∞, respectively.

3.2 Virtual Party MPC

Party $\mathcal{P}_i \in \mathcal{P}_\infty$ | "Offline" Party $\mathcal{P}_0$

Retrieve $(\vec{B}_{i',j,t})_{i',j,t}$ and $(H_{j,t}, k_{i,j,t})_{j,t}$ and do:

Initialize: (Figure 11)

Run init MPC protocol on $((\vec{B}_{i',j,t})_{i'}; (k_{i,j,t}))_{j,t}$

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

Prepare Payload:

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

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

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

Aggregate-Prove: (Figure 5)

$$\psi_{j,t} \leftarrow \Theta_{R_{j,t}}((\vec{B}_{i,j,t}, H_{j,t}); (k_{i,j,t})_j)$$

$$(S_j)_j, \psi' \leftarrow \Theta'(((\vec{V}_{i,j,t}, \vec{Z}_{i,j,t})_t, S_{i,j})_j; (\zeta_{i,j,t}, \mu_{i,j,t})_j)$$

for $\vec{V}_{i,j,t} = \vec{B}_{i,j,t}^{r_{j,t}}$ and $\vec{Z}_{i,j,t} = \vec{B}_{i,j,t}^{m} \cdot \vec{U}_{i,j,t}^{r_{j,t}}$

$$(\vec{U}_{j,t})_{j,t}, \psi'' \leftarrow \Theta''((\vec{B}_{i,j,t}, \vec{U}_{i,j,t})_{j,t}; (k_{i,j,t})_{j,t})$$

$$\xrightarrow{\psi', \psi'', ((r_{j,t}, U_{j,t}, \psi_{j,t})_t, S_j)_j}$$

Same as Figure 9.

Figure 10: Threshold ECDSA: MPC Signing (Σ_{sign})

3.2.1 `init` MPC Protocol (i.e. CMP)

Definition 3.6. 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 $\boldsymbol{E} = \boldsymbol{I}(2^\ell)$

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

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

Definition 3.7. 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 $\boldsymbol{E} = \boldsymbol{I}(2^\ell)$

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

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

$$\gamma_i \leftarrow \mathbb{F}_q,\ \vec{\Gamma}_i = \mathsf{com}(\gamma_i, Y)$$
$$K_i = \mathsf{enc}_i(k_i) \text{ and } G_i = \mathsf{enc}_i(\gamma_i)$$
$$\forall j, \psi_{j,i} \leftarrow \Xi^1_{j,i}((K_i, \vec{B}_i, G_i, \vec{\Gamma}_i); (k_i, \gamma_i))$$

$$\underset{\psi_{i,j}, K_j, G_j, \vec{\Gamma}_j}{\overset{\psi_{j,i}, K_i, G_i, \vec{\Gamma}_i}{\rightleftarrows}}$$

$$\forall j, \text{ sample } \beta_{i,j}, \hat{\beta}_{i,j} \leftarrow \boldsymbol{I}(2^{2\ell+\varepsilon}) \text{ and do:}$$
$$\text{Set } \hat{D}_{j,i} = [K_j^{\gamma_i} \cdot \mathsf{enc}_j(\hat{\beta}_{i,j})]_{N_j^2} \text{ and } \hat{F}_{i,j} = \mathsf{enc}_i(\hat{\beta}_{i,j})$$
$$D_{j,i} = [K_j^{x_i} \cdot \mathsf{enc}_j(\beta_{i,j})]_{N_j^2} \text{ and } F_{i,j} = \mathsf{enc}_i(\beta_{i,j}) \text{ and } H_i = H^{\gamma_i}$$
$$\phi_{j,i} \leftarrow \Xi^2_{j,i}((H_i, X_i, \vec{\Gamma}_i, D_{j,i}, F_{i,j}, \hat{D}_{j,i}, \hat{F}_{i,j}); (x_i, \gamma_i, \beta_{i,j}, \hat{\beta}_{i,j}))$$

$$\underset{H_j, D_{i,j}, F_{j,i}, \hat{D}_{i,j}, \hat{F}_{j,i}, \phi_{i,j}}{\overset{H_i, D_{j,i}, F_{i,j}, \hat{D}_{j,i}, \hat{F}_{i,j}, \phi_{j,i}}{\rightleftarrows}}$$

$$\text{Set } \delta_i = [k_i\gamma_i + \textstyle\sum_{j\neq i} \mathsf{dec}_i(\hat{D}_{i,j}) + \hat{\beta}_{i,j}]_q$$
$$\text{Set } \Lambda = \textstyle\prod_j H_j \text{ and } \Delta_i = \Lambda^{k_i}$$
$$\omega_i \leftarrow \Xi^3_\Lambda(\Delta_i, \vec{B}_i; k_i)$$

$$\underset{\delta_j, \Delta_j, \omega_j}{\overset{\delta_i, \Delta_i, \omega_i}{\rightleftarrows}}$$

$$\text{Set } \delta = \textstyle\sum_j \delta_j,\ \Delta = \prod_j \Delta_j \text{ and check } g^\delta = \Delta$$
$$\text{Set } R = H^{\delta^{-1}} \text{ and } \chi_i = [k_i x_i + \textstyle\sum_{j\neq i} \mathsf{dec}_i(D_{i,j}) + \beta_{i,j}]_q$$
$$\text{Set } r = R|_{\text{x-axis}} \text{ and output } (R, \chi_i, r)$$

Figure 11: Threshold ECDSA: Virtual MPC (init)

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

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

Definition 3.8. Define Schnorr protocol Ξ^3_Λ for $(\phi, \boldsymbol{E})$ where $\boldsymbol{E} = \boldsymbol{I}(2^\ell)$ and

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

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 4.3 and 4.5.

4.1 Unforgeability & Simulatability imply UC Security

Let Sig denote a signature scheme and let Σ be a threshold protocol for Sig. We show that if Σ and Sig 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 which essentially corresponds to the standalone definition of security.

Let $\mathcal{A}$ denote an adaptive adversary and write $\text{Real}_{\mathcal{A}}$ for the adversary's view in an execution of Σ in the presence of an *adaptive* PPTM adversary $\mathcal{A}$. Recall that $\mathcal{H}$ denotes the random oracle. Without loss of generality assume that $\text{Real}_{\mathcal{A}} = (\mathsf{pk}^{\Sigma}, \ldots)$, where pk^{Σ} denotes the public key resulting from the execution of 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 $\text{Ideal}_{\mathcal{S}} = (\mathsf{pk}^{\mathcal{G}}, \text{Out}^{\mathcal{S}})$ for the pair of random variable consisting of the public key generated by $\mathcal{G}$ and the simulator's output.

Definition 4.2 (Simulatability)**.** Using the notation above, we say that Σ is $\mathcal{G}$-simulatable in the ROM if the following holds for every adversary $\mathcal{A}$. 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, (2) if $\mathcal{A}$ does not corrupt all parties in some $\boldsymbol{Q} \in \mathbb{Q}$ simultaneously in any given epoch, then $\{\text{Real}_{\mathcal{A}}\} \equiv \{\text{Ideal}_{\mathcal{S}}\}$.

Theorem 4.3. *Let* Sig *denote a signature scheme and let* Σ *denote a threshold-*Sig *protocol. Let* $\mathcal{G}$ *denote an augmented signature oracle such that*

- Sig *is* $\mathcal{G}$*-existentially unforgeable.*
- Σ *is* $\mathcal{G}$*-simulatable in the ROM.*

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

The above theorem informally states that if Σ achieves standalone security in the $\mathcal{G}$-hybrid model, then Σ achieves UC security (assuming that $\mathcal{G}$ is not useful for forging signatures).

Proof. First, we describe the UC simulation, which is trivial. The simulator simply runs the code of the honest parties. Furthermore, every time the honest parties output a signature, then the simulator submits the resulting signature-string to the functionality, and, depending on the adversary's corruption pattern and the protocol's key-refresh schedule, the simulator registers parties as corrupted and/or quarantined. It is not hard to see that the environment $\mathcal{Z}$ can distinguish real from ideal execution only if it can forge signatures in the protocol (i.e. in the real world). However, since Σ is $\mathcal{G}$-simulatable, it follows by Definition 4.2 that the interaction between $\mathcal{Z}$ and the honest parties can be simulated using the oracle to $\mathcal{G}$, which in turn implies that $\mathcal{G}$ is useful for forging signatures of Sig, in contradiction with the hypothesis of the theorem. $\square$

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. [9] 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 [26] 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 4.4. We say that strong RSA, DDH and DCR hold true if there exists a negligible function $\nu(\cdot)$ such that for every PPTM $\mathcal{A}$ 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 $z = \beta$

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 $(\mathsf{sign}, \mathrm{msg}, R)$, 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 $m = \mathcal{F}(\mathrm{msg})$ 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

Theorem 4.5. *Assuming DDH, DCR, and strong RSA, it holds that Σ_{ecdsa} is $\mathcal{G}^*$-simulatable.*

Proof. At the beginning of simulation, the simulator chooses a random non-corrupted party, which is called the special party, and all other non-corrupted parties are simulated by running their code as prescribed. To deal with adaptive corruptions, we assume that the special party is chosen afresh every time Σ_{refr} is simulated (the simulation is reset, via rewinding, to the last key refresh if the adversary decides to corrupt the special party). Furthermore, $\mathcal{S}$ simulates the random oracle as well (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)
2. Extract $\mathcal{A}$'s Paillier keys and ECDSA key shares by rewinding.
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.

Effectively, $\mathcal{S}$ does not use the messages intended for the special party (unless the simulation is terminated early, if, say, $\mathcal{A}$ sends a faulty proof). See Figure 13 for the full description.

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:

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

It is not too hard to see that that the second experiment is identical to the real execution, as long as all the proofs are sound and the simulator extracts the right values (which follows from Theorem 5.1 and Fact 5.10 under strong RSA). Next, notice that the the first experiment is identically-distributed to the simulated

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_{refr}) Reassign $\mathcal{P}_b \leftarrow \boldsymbol{H} = \boldsymbol{P} \setminus \boldsymbol{C}$ and do:
  1. For $\hat{\beta} \leftarrow \mathbb{F}_q$, set $\hat{X}_{\boldsymbol{H}} = g^{\hat{\alpha} + \hat{\beta}} \cdot \prod_{i \in \boldsymbol{H}} X_i$ with $\hat{\alpha} \leftarrow \mathbb{F}_q$ if $\mathcal{P}_0 \notin \boldsymbol{C}$ and $\hat{\alpha} = 0$ otherwise.
  2. Sample $\{\hat{x}_i \leftarrow \mathbb{F}_q\}_{i \in \boldsymbol{H} \setminus \{b\}}$ and set $\hat{X}_i = g^{\hat{x}_i}$ and $\hat{X}_b = \hat{X}_{\boldsymbol{H}} \cdot (\prod_{i \in \boldsymbol{H} \setminus b} \hat{X}_i)^{-1}$.
  3. Run the simulation for Σ_{kgen} (with $\hat{X}_0$ if $\mathcal{P}_0 \notin \boldsymbol{C}$).
  4. When obtaining α from $\mathcal{A}$, if $\mathcal{P}_0 \in \boldsymbol{C}$, set $\hat{\alpha} = \alpha$.
     Hand over $(\hat{X}_i)_{i \in \boldsymbol{H}}$ and random $(\beta_j)_{j \in \boldsymbol{C} \setminus \{0\}}$ to $\mathcal{A}$ s.t. $\sum_{j \in \boldsymbol{C} \setminus \{0\}} \beta_j = \hat{\alpha} - \hat{\beta}$.
- (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. Run the simulation for Σ_{cmp} (Figure 14). Obtain output $(\chi^*_{j,\ell})_{j,\ell}$.
     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 $\eta_{b,j,\ell} \leftarrow \boldsymbol{I}(2^\varepsilon)$ and $\vec{U}_{j,\ell} \leftarrow \mathbb{G}^2$
  3. Run the simulation for proof aggregation for Θ, Θ' and Θ'' (Claim 2.13).
     When obtaining $\{\vec{U}_{i,j,\ell} = (U_{i,j,\ell}, U'_{i,j,\ell})\}_{i \neq b}$, set $\vec{U}_{b,j,\ell} = \vec{U}_{j,\ell} \cdot \prod_{i \neq b}(U_{i,j,\ell}, U^y_{i,j,\ell} \cdot g^{\chi_{i,j,\ell}})^{-1}$.
     Simulate ψ'' according to $\vec{U}^*_{b,j,\ell} = \vec{U}_{b,j,\ell} \cdot (\mathbb{1}, X_\infty^{-\sum_{i \neq b} k_{i,j,\ell}} \cdot g^{\sum_{i \neq b} \chi_{i,j,\ell}})$

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

FIGURE 14 (Simulator for Σ_{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 $k_j$, $\gamma_j$ and do:
  1. Set $\{F_{b,j}, \hat{F}_{b,j} = \mathsf{enc}_b(0)\}_{j \neq b}$ and $\{D_{b,j} = \mathsf{enc}_b(\alpha_j), \hat{D}_{b,j} = \mathsf{enc}_j(\hat{\alpha}_j)\}_{j \neq b}$ for $\alpha_{j,b}, \hat{\alpha}_{j,b} \in \boldsymbol{I}(2^{2\ell+\varepsilon})$
  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 $(H_j, \ldots, \phi_{b,j})_{j \neq b}$, $\gamma_j$ and do:
  1. Set $\delta_b = [\delta - \sum_{j \neq b} \hat{\alpha}_j - (\sum_{i,j \neq b} k_i \gamma_j)]_q$ and and $\Delta_b = g^{\delta} \cdot \prod_{j \neq b} \Lambda^{-k_i}$.
  2. Calculate $\omega_b$ 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 $\chi^* = [\sum_{j \neq b} \alpha_j + (\sum_{i,j \neq b} k_i x_j)]$
```

Figure 14: Simulator for Σ_{cmp}

experiment. To conclude, we note that 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). We note that the simulation concludes with overwhelming probability in time $n \log(n) \cdot \mathsf{time}_{\Sigma}$ where time_{Σ} is the running time of Σ_{ecdsa} (because the simulator is required to guess the identity of the simulated honest party). This concludes the proof. □

5 Proof of Soundness for Batch-Proving

In this section we prove a general claim (Theorem 5.1) about the soundness of an m-batched Schnorr protocol depending on the soundness of the underlying non-batched protocol and some additional technical requirements. Then, in Section 5.2 we show that our batched protocols from Section 3 satisfy the hypothesis of Theorem 5.1 for suitable choice of parameters.

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

Then, for $\beta, \gamma \in \mathsf{negl}(\kappa)$, it holds that Π^* is (μ^*, ν^*)-secure for

$$\begin{cases} \mu^* = m \cdot \mu \\ \nu^* = \delta + \varepsilon + \mathsf{negl}(\kappa) \end{cases}.$$

Before we prove Theorem 5.1 we point the reader to the relevant claims for showing that the batched Schnorr protocols from Section 3 are sound. For $\beta, \gamma, \delta, \varepsilon$ as in Theorem 5.1, it holds that $\varepsilon \in \mathsf{negl}(\kappa)$ by Theorem 5.4 for $\boldsymbol{V}$ explicitly defined in Section 5.2, and $\beta \in \mathsf{negl}(\kappa)$ by Fact 5.8 since $\log(q) = |\mathbb{G}| \sim \kappa$ and the Paillier moduli $\{N_i\}_{i=1}^n$ are large, squarefree and do not admit small factors (Guaranteed by protocol Φ from round 3 of Σ_{kgen}). Finally, $\gamma, \delta \in \mathsf{negl}(\kappa)$ by Fact 5.9 and Fact 5.10 respectively.

5.1 Proof of Theorem 5.1

First, we show HVZK. Write σ_i for the random variable calculated as $\sigma_i = \alpha + \sum_{j=1}^{i} e_j w_j$ for $\vec{e} \leftarrow \boldsymbol{E}^i$ and note that $\mathrm{SD}(\sigma_0, \sigma_1) \leq \mu$ by the HVZK property of the underlying protocol. Further observe that $\mathrm{SD}(\sigma_0, \sigma_m) = \mathrm{SD}(\sigma_0, \sigma_{m-1} + e_m w_m) \leq \mathrm{SD}(\sigma_0, \sigma_{m-1}) + \mathrm{SD}(\sigma_0, \sigma_0 + e_m w_m)$ and the HVZK part of the claim follows by simple induction. Hereafter, let $\mathcal{A}$ denote an adversary that breaks soundness with probability λ and fix $r \in \mathsf{poly}$ such that $\lambda r \in \omega(\log(\kappa))$ and $m/\lambda r \in o(1)$. Consider the following experiment.

Experiment 5.2. Define $\mathcal{E}$ 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 m 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 5.3. *$\mathcal{E}$ outputs 0 in Experiment 5.2 with probability at most $\beta \cdot rm + \mathsf{negl}(\kappa)$.*

Proof. By Chernoff, since $\lambda r \in \omega(\log(\kappa))$ and $m/\lambda r \in o(1)$, then, with overwhelming probability, there are at least $m+2$ transcripts. Furthermore, the probability that $(\vec{f}_1, \ldots, \vec{f}_{m+1}) \notin \boldsymbol{V}$ is at most $\beta \cdot rm$, by union bound. □

Assuming that, for some $j \in [m]$, $X_j \neq \phi(w_j)$ for all $w_j \in \boldsymbol{S}$, consider the following sequence of events.

1. Run Experiment 5.2 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 $\tau = (A, \vec{e}, z)$ denote the last (unused) transcript.
3. $\hat{z} = \alpha + \sum_{j=1}^m e_j w_j \notin \boldsymbol{S}$.
4. $\hat{z} = z$ and $\phi(\hat{z}) = \phi(z)$.

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

$$\underbrace{\beta \cdot mr + \mathsf{negl}(\kappa)}_{\text{Item 1}} + \underbrace{\varepsilon + r \cdot \gamma + \delta}_{\text{Items 2, 3, 4}}$$

□

5.2 Putting Everything Together

In this section we prove auxiliary claims for showing our batched 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

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

Let Π denote the m-batched 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

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

is invertible over $\mathbb{F}_q$ and $(\mathbb{Z}_{N_i}, +, \cdot)$, for all $i \in [n]$. In the remainder, we state and prove Theorem 5.4 (Extractability) and Facts 5.8 (Robustness), 5.9 (Unpredictability) and 5.10 (Binding).

5.2.1 Extractability

Theorem 5.4. *Under the strong-RSA assumption, for $\varepsilon \in \mathsf{negl}(\kappa)$ and $(\hat{N}, t, s_1, \ldots) \leftarrow \mathsf{ped}(1^\kappa)$, it holds that Π satisfies $(\varepsilon, \boldsymbol{V})$-extractability.*

Proof. For strong-RSA challenge $(N, c) \leftarrow \mathsf{sRSA}(1^\kappa)$, the Pedersen parameters $(\hat{N}, t, s_1, \ldots)$ are set as[17] $(\hat{N}, t) = (N, c)$ 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 $t \in \mathbb{Z}_N^*$.[18] 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[19]

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

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.

$$\forall i, \quad \begin{cases} \mu_i^{N_0} \prod_{k=1}^r A_k^{z_{i,k}} = D \cdot \prod_{j=1}^m C_j^{e_{i,j}} \mod N_0 \\ t^{\gamma_i} \prod_{k=1}^r s_k^{z_{i,k}} = T \cdot \prod_{j=1}^m S_j^{e_{i,j}} \mod \hat{N} \end{cases} \tag{2}$$

and assume that $\{\vec{e}_i\}_{i=1}^{m+1} \in \boldsymbol{V}$. Define the square integer matrix E as in Equation (1) 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

$$t^{\Delta_\gamma} \cdot \prod_{k=1}^r s_k^{\Delta_z^{(k)}} = R^{\Delta_e} \mod \hat{N} \text{ s.t. } \begin{cases} R = S_i & \text{if } i \neq m+1 \\ R = T & \text{otherwise} \end{cases} \tag{3}$$

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

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

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

[19] The general case follows straightforwardly.

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[20] of $\hat{N}$ and thus to prove our theorem it suffices to bound the probability that Δ_e does not divide one of $\{\Delta_z^{(k)}\}_{k=1}^r$ or Δ_γ. Let $\hat{\varepsilon} = \Pr[\Delta_e \not| \Delta_\gamma \vee \Delta_e \not| \Delta_z^{(1)} \vee \ldots \vee \Delta_e \not| \Delta_z^{(r)}]$ 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 $\hat{\varepsilon} \leq \Pr[\Delta_e \not| \Delta_\Sigma] + \sum_{k=1}^r \Pr[\Delta_e \not| \Delta_z^{(k)} \wedge \Delta_e \mid \Delta_\Sigma]$. Apply Claim 5.5, Claim 5.6 and Claim 5.7 and conclude that, since $i \in [m+1]$ was chosen arbitrarily,

$$\varepsilon \leq (m+1) \cdot \hat{\varepsilon} + \mathsf{negl}(\kappa) \in \mathsf{negl}(\kappa). \tag{4}$$

Claim 5.5. *It holds that* $\Pr[\Delta_e \not| \Delta_\Sigma] \in \mathsf{negl}(\kappa)$*,*

Proof. Let $d = \gcd(\Delta_e, \Delta_\Sigma)$. If $\Delta_e \not| \Delta_\Sigma$, 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 $t = t^{u \cdot d_e + v \cdot d_\Sigma} = t^{u \cdot d_e} \cdot R^{v \cdot d_e} = (t^u \cdot R^v)^{d_e} = m^{d_e} \mod N$. □

Next, we prove that if $\Delta_e \not| \Delta_z^{(j)}$ for some $j \in r$, then the probability that $\Delta_e \mid \Delta_\Sigma$ is bounded away from 1 (together with Claim 5.5, this yields Equation (4)).

Claim 5.6. $\Pr\left[\Delta_e \mid \Delta_\Sigma \;\middle|\; \Delta_e \not| \Delta_z^{(j)}\right] \leq 1/2 + \mathsf{negl}(\kappa)$*, 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 $\Delta_e \not| Q \cdot \Delta_z^{(j)}$, then there exists a prime power a^b such that

$$\begin{cases} a^b \mid Q \cdot \Delta_z^{(j)} \\ a^{b+1} \not| Q \cdot \Delta_z^{(j)} \\ a^{b+1} \mid \Delta_e \end{cases}$$

Finally, notice that if $\Delta_e \not| Q \cdot \Delta_z^{(j)}$ 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,

$$\Pr\left[\Delta_e \mid \Delta_\Sigma \;\middle|\; \Delta_e \not| \Delta_z^{(j)}\right] \leq \Pr[\gcd(\Delta_e, Q) \neq 1] + \Pr\left[\Delta_e \mid \Delta_\Sigma \;\middle|\; \Delta_e \not| \Delta_z^{(j)} \wedge \gcd(\Delta_e, Q) = 1\right]$$

and $\Pr\left[\Delta_e \mid \Delta_\Sigma \;\middle|\; \Delta_e \not| \Delta_z^{(j)} \wedge \gcd(\Delta_e, Q) = 1\right] \leq 1/a$, 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 (2) , 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 5.7 to extract the randomizers (the ν's).

[20] Factoring is reducible to strong RSA.

Claim 5.7. *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 5.4. □

5.2.2 Robustness, Unpredictability and Binding

Fact 5.8 (Robustness)**.** *Let $\ell, p \in \mathbb{Z}$ such that $\ell, \log(p) \sim \kappa$. 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*

$$\Pr_{\vec{e}_{j+1} \leftarrow (\pm 2^\ell)^m} [(\vec{e}_{j+1}, 1) \in \langle (\vec{e}_1, 1), \ldots, (\vec{e}_j, 1) \rangle_{\mathbb{Z}_p}] \in \mathsf{negl}(\kappa).$$

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

Fact 5.9 (Unpredictability)**.** *Let $\vec{w} \in \mathbb{Z}^m$ and $\ell, \varepsilon \in \mathbb{Z}$ with $\varepsilon, \ell \sim \kappa$. For any $\alpha \in \mathbb{Z}$, if $\vec{w} \notin \boldsymbol{J}(N \cdot 2^\ell)^m$ then*

$$\Pr_{\vec{e} \leftarrow (\pm 2^\ell)^m} [\alpha + \sum_j e_j w_j \in \boldsymbol{J}(N \cdot 2^\ell)] \in \mathsf{negl}(\kappa).$$

Proof. For any fixed $\alpha \in \mathbb{Z}$, for any $|w| > N \cdot 2^{\ell+\varepsilon}$, it holds that

$$\begin{aligned} \Pr_{e \leftarrow \pm 2^\ell} [\alpha + we \in \pm N \cdot 2^{\ell+\varepsilon}] &= \Pr[ew \in -\alpha \pm N \cdot 2^{\ell+\varepsilon}] \\ &\leq \Pr[e \in \lceil -\alpha/w \rfloor \pm 1] \\ &\leq \frac{3}{2^\ell} \end{aligned}$$

□

Fact 5.10 (Pedersen Binding)**.** *Under the factoring assumption, for $\pi = (\hat{N}, t, \vec{s}, \ldots) \leftarrow \mathsf{ped}(1^\kappa)$, for any efficient $\mathcal{A}$, 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 5.4, 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] 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.

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

[10] R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. Cryptology ePrint Archive, Report 2021/060, 2021. https://eprint.iacr.org/2021/060.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] J. Doerner, Y. Kondi, E. Lee, and a. shelat. Secure two-party threshold ECDSA from ECDSA assumptions. In *2018 IEEE Symposium on Security and Privacy*, pages 980–997. IEEE Computer Society Press, May 2018. doi: 10.1109/SP.2018.00036.

[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] 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.

[22] R. Gennaro and S. Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *ACM CCS 2018*, pages 1179–1194. ACM Press, Oct. 2018. doi: 10.1145/3243734.3243859.

[23] 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.

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

[25] 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.

[26] 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.

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

[28] 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.

[29] Y. Lindell. Fast secure two-party ECDSA signing. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, pages 613–644, 2017.

[30] 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.

[31] 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.

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

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

[34] 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.

[35] 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 15 (Ideal Threshold Signature Functionality $\mathcal{F}_{\mathsf{tsig}}$)

Key-generation:

1. Upon receiving $(\texttt{keygen}, \mathit{ssid})$ from some party $\mathcal{P}_i$, interpret $\mathit{ssid} = (\ldots, \boldsymbol{P}, \mathbb{Q})$, where $\boldsymbol{P} = (\mathcal{P}_1, \ldots, \mathcal{P}_n)$.
   - If $\mathcal{P}_i \in \boldsymbol{P}$, send to $\mathcal{S}$ and record $(\texttt{keygen}, \mathit{ssid}, \mathcal{P}_i)$.
   - Otherwise ignore the message.
2. Once $(\texttt{keygen}, \mathit{ssid}, j)$ 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 $\boldsymbol{Q} \in \mathbb{Q}$, 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 $\boldsymbol{Q} \in \mathbb{Q}$ 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 $\boldsymbol{Q} \in \mathbb{Q}$ 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 $\boldsymbol{Q} \in \mathbb{Q}$ 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 15: Ideal Threshold Signature Functionality $\mathcal{F}_{\mathsf{tsig}}$

A Missing ZK Protocols/Proofs

A.1 Security Proof for Multi-Pedersen Membership

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

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

Claim A.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. □

A.2 Well-Formed Modulus & Ciphertext ZK Proof

Next we describe the missing ZK-proof $\Phi_{(\cdot)}$ [9] from Section 3.1.1. The proof is a combination of two Schnorr protocols $\Pi_{(\cdot)}$ and $\Theta_{(\cdot)}$ ("tight range proof" from [14] and "no small-factors proof" from [9]), and the ZK protocol Ξ ("Paillier-Blum proof" from [9]), all described below. Soundness and HVZK follow from the soundness and HVZK of the underlying protocols and Theorem 5.1. Recall that $\boldsymbol{I}(M) = \pm M$ and $\boldsymbol{J}(M) = \pm M \cdot 2^\varepsilon$. 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 16. For the security properties (HVZK, soundness & Extraction) of Ξ, we refer the reader to [9, p. 28].

FIGURE 16 (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 16: Paillier-Blum Modulus ZK – $\Xi(N; p, q)$

Tight Range. Define Π_σ for $(\phi, \boldsymbol{E})$ for $\boldsymbol{E} = \boldsymbol{I}(2^\ell)$, $\sigma = (\pi, T, \vec{A}, N)$ 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 $x, \mu, \alpha_i, \rho_i \in \boldsymbol{I}(\hat{N} \cdot 2^\ell)$, $\beta \in \boldsymbol{I}(\hat{N} \cdot 2^{2\ell})$ 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 $\boldsymbol{E} = \boldsymbol{I}(2^\ell)$ 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 $p, q \in \boldsymbol{I}(\sqrt{N} \cdot 2^\ell)$, $u, v \in \boldsymbol{I}(\hat{N} \cdot 2^\ell)$ and $w \in \boldsymbol{I}(2^{2\ell} \cdot \hat{N}\sqrt{N})$. Define $\boldsymbol{S}$ analogously with $\boldsymbol{J}(\cdot)$.

A.2.1 ZK Proof Description

Define Φ_π for $\pi = (\hat{N}, s, t, \ldots)$ in Figure 17.

Prover

Common Input: N, W, X, ℓ

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 $p, q \in \boldsymbol{I}(2^\ell\sqrt{N})$ and $x \in [0, 2^\ell]$.

1. Find $\alpha_1, \alpha_2, \alpha_3$ such that $4(2^\ell - x) \cdot x + 1 = \alpha_1^2 + \alpha_2^2 + \alpha_3^2$
2. Sample $(\rho_i \leftarrow \boldsymbol{I}(2^\ell \cdot \hat{N}))_{i=1,2,3}$ and set $\{A_i = s^{\alpha_i} t^{\rho_i} \mod \hat{N}\}_{i=1,2,3}$
3. Sample $\mu, u, v \leftarrow \boldsymbol{I}(2^\ell \cdot \hat{N})$ and set $(S, T) = [(s^x t^\mu, (s^{2^\ell} \cdot S^{-1})^4)]_{\hat{N}}$ 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, ℓ.

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 $T = [(s^{2^\ell} \cdot S^{-1})^4]_{\hat{N}}$ 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 17: Well-Formed Modulus & Ciphertext Φ_π

B 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 and key refresh as one-time or sporadic costs which do 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. 6), it is possible to infer real-world communication costs.

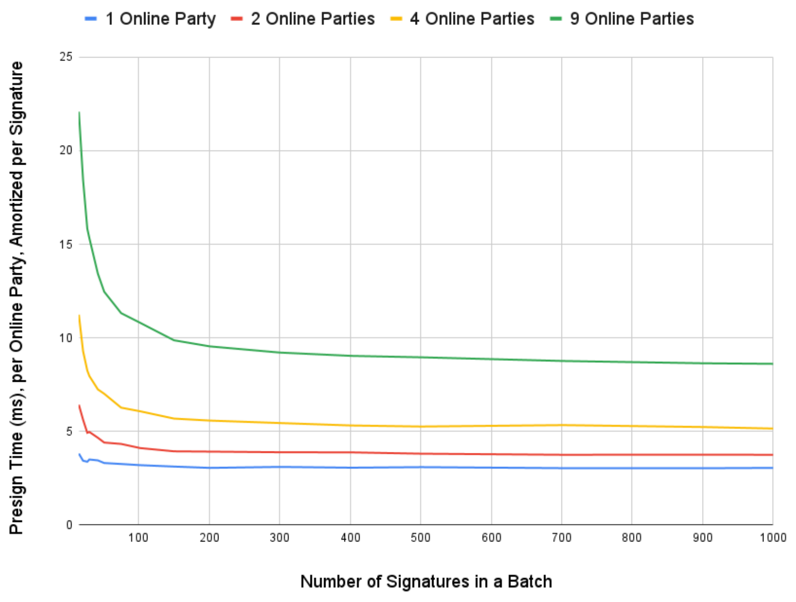


Figure 18: (Plot 1) CPU time for presigning for each online party, viewed as a function of the batching parameter, i.e. number of future signatures in the batch. Reported values are amortized over the number of future signatures (so total costs scale linearly with respect to this quantity). We ran four 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.

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.[21] 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 each online party and *Plot 2:* for the offline party as a function of the batching parameter, i.e. how many presignatures are handled in a single batch. *Plot 3:* Computation time per message during signing for each online party, as a function of the number of parties. *Plot 4:* 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 5:* Performance improvement (speedup) when using the packing optimization vs not using it during presigning, as a function of the number of parties.

[21]https://github.com/udi0peled/asymmetric_offline_cmp (accessed November 2022).

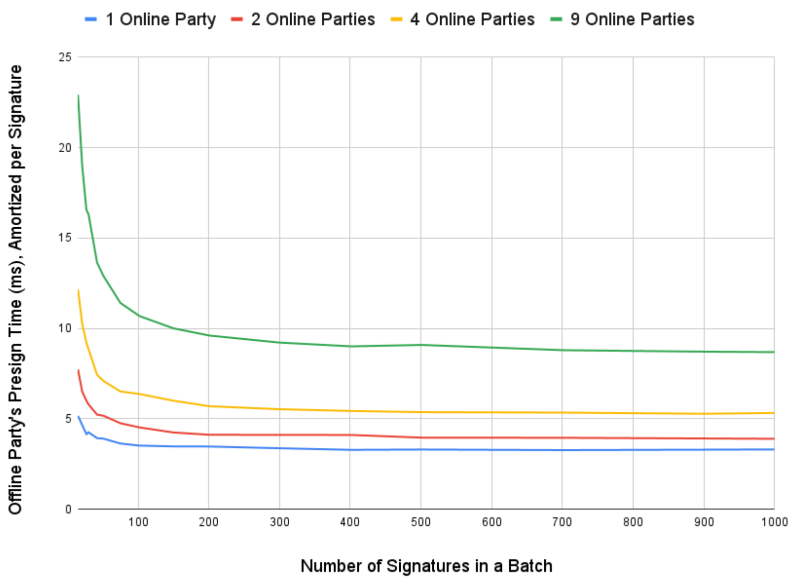


Figure 19: (Plot 2) Same measurements as Plot 1, but for the offline party.

Choice of Parameters & Machine Specs. We instantiate the random oracle with SHA-512 (for Fiat-Shamir, commitments, etc. . .) and we opted to instantiate ECDSA 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.

B.1 Concluding Remarks

Plots 1 & 2. 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 3. 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 4. 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.

Plot 5. As mentioned in the caption of plot 5, 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

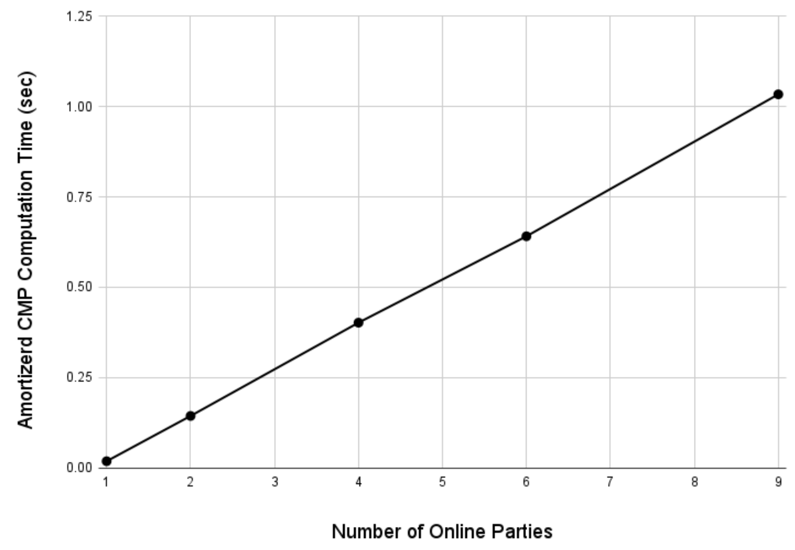


Figure 20: (Plot 3) Total CPU time for the online-signing phase, i.e. *init/CMP* + aggregation, for each online party (there is no offline party in this phase), per signature. The reported values were calculated by running the protocol 100 times and taking the average. We note that the bulk of the online-signing 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).

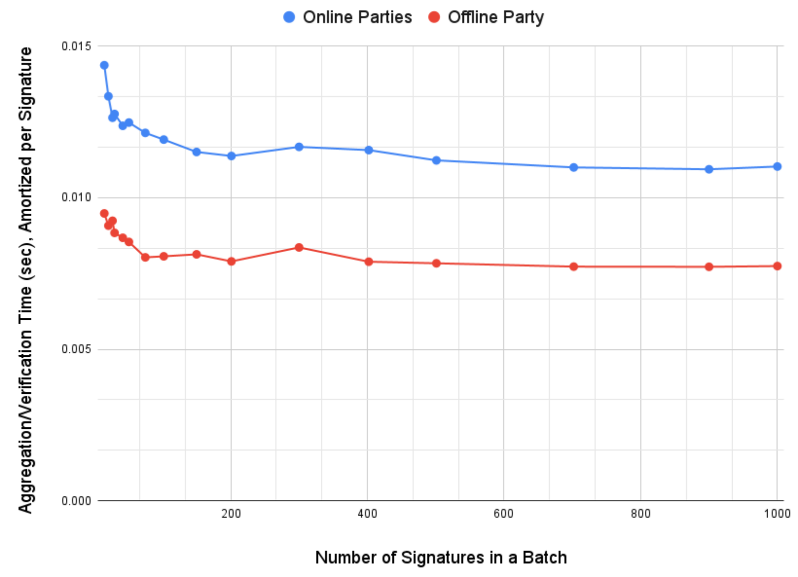


Figure 21: (Plot 4) CPU time for aggregation process for each online party and CPU running time for verification/finalization process for the offline party, viewed as a function of the batching parameter amortized over the number of signatures. We note that the displayed costs are not affected by the number of online parties (though there is a theoretical dependency for the blue line, cf. concluding remarks).

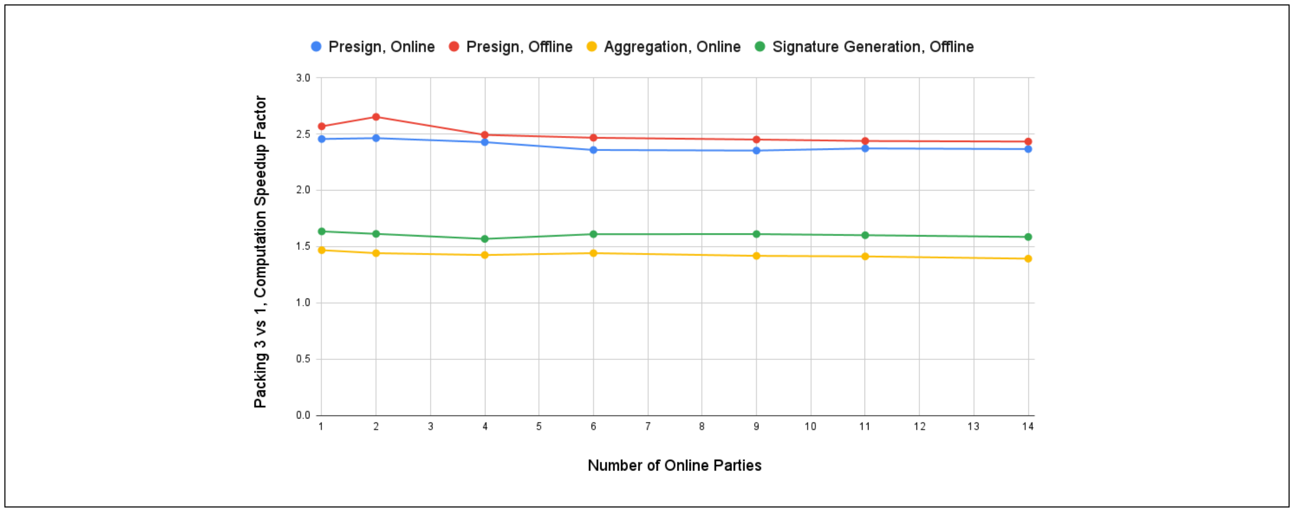


Figure 22: (Plot 5) CPU time speedup for presigning when packing three plaintexts into one Paillier ciphertext, compared to no packing at all, as a function of the number of online parties. E.g. when using packing number 3 during presigning, the parties run roughly 2.5 times faster compared to packing 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).

the most expensive component of our protocol). However, it may yet be desirable to increase the packing number if the communication benefit outweighs the computational slowdown.