Real Threshold ECDSA

Harry W. H. Wong, Jack P. K. Ma, Hoover H. F. Yin, and Sherman S. M. Chow*
Department of Information Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong

***Abstract*—Threshold ECDSA recently regained popularity due to decentralized applications such as DNSSEC and cryptocurrency asset custody. Latest (communication-optimizing) schemes often assume all n or at least $n' \geq t$ participating users remain honest throughout the pre-signing phase, essentially degenerating to n'-out-of-n' multiparty signing instead of t-out-of-n threshold signing. When anyone misbehaves, all signers must restart from scratch, rendering prior computation and communication in vain. This hampers the adoption of threshold ECDSA in time-critical situations and confines its use to a small signing committee.**

To mitigate such denial-of-service vulnerabilities prevalent in state-of-the-art, we propose a robust threshold ECDSA scheme that achieves the t-out-of-n threshold flexibility "for real" throughout the whole pre-signing and signing phases without assuming an honest majority. Our scheme is desirable when computational resources are scarce and in a decentralized setting where faults are easier to be induced. Our design features 4-round pre-signing, $\mathcal{O}(n)$ cheating identification, and self-healing machinery over distributive shares. Prior arts mandate abort after an $\mathcal{O}(n^2)$-cost identification, albeit with 3-round pre-signing (Canetti et al., CCS '20), or $\mathcal{O}(n)$ using 6 rounds (Castagnos et al., TCS '23). Empirically, our scheme saves up to $\sim$30% of the communication cost, depending on at which stage the fault occurred.

I. Introduction

Signature is mission-critical for ensuring the authenticity of disseminated information and authorization, such as records from a domain name system (DNS) or transactions. Meanwhile, availability is no less important. Combining both security features requires a t-out-of-n threshold signature scheme, which (t, n)-secret-shares private signing key to n parties – any t out of n signers can sign without reconstructing the private key to ensure availability, while forgery would be impossible even if a $(t-1)$-adversary [3] compromised $(t-1)$ signers.

Cryptocurrencies, blockchain, or distributed systems often employ (n, n) multisignatures [22] or (t, n) threshold signatures. Elliptic curve digital signature algorithm (ECDSA) [20] is a popular choice for its small signature size. It also features very efficient online signing, given a *pre-signature* prepared by a pre-signing protocol in an "offline" phase before knowing the message to sign. Threshold ECDSA is welcomed by high-stake decentralized applications like decentralized autonomous organization, exchange, and mixing services. A flurry of new results in threshold signatures appeared in recent years.

*Corresponding author. Sherman Chow is supported by the General Research Fund (CUHK 14210621) from Research Grant Council, Hong Kong.

Network and Distributed System Security (NDSS) Symposium 2023
27 February - 3 March 2023, San Diego, CA, USA
ISBN 1-891562-83-5
https://dx.doi.org/10.14722/ndss.2023.24817
www.ndss-symposium.org

A. A Neglected Problem

A *robust* protocol requires an *honest majority* ($n \geq 2t-1$), where any $(t-1)$-adversary cannot prevent share holders from signing. Optimizing for performance, most threshold ECDSA schemes [13], [21], [6], [11], [4], [7] assume an ideal case with barely sufficient (t) signers participating honestly in the pre-signing phase (or earlier steps in signing for schemes with no online/offline signing explicated [14], [3]) and remaining available until the online signing phase concludes. Starting the (pre-)signing protocol with $n' \geq t$ signers does not help these schemes since the results can only be used by *exactly* the same set of parties who generated them, not even tolerating one signer less, degenerating to n'-out-of-n' non-threshold signing.

Assuming all or at least $n' \geq t$ signers faithfully computed the pre-signatures, Gągol *et al.* [18], Damgård *et al.* [10], and Pettit [24] consider robustness for *online* signing, while only Gągol *et al.* [18] consider a dishonest majority setting, *i.e.*, $n < 2t-1$. (No protocol can achieve robustness in the dishonest-majority case.) These form a step forward, but the core problem still exists in pre-signing and remains unsolved.

Buying time through such a seemingly-benign vulnerability against denial-of-service (DoS) attacks can have avalanche consequences. In a decentralized setting, secret-share holders (*e.g.*, servers of DNS security extensions, stakeholders of an organization, or shareholders of a cryptocurrency asset) can be geographically distributed; their timely responses are often not guaranteed. Unfortunately, the authorization power attached to the signature (*e.g.*, strategic/investment decisions) can be time-critical. Any delay gives a golden time window to be exploited (*e.g.*, DNS poisoning). Also, faulty behavior is easier to be induced in a decentralized setting but may not be identifiable in existing schemes. An honest signer may become faulty due to low connectivity, running an older version of the software/protocol, processing a slightly out-of-sync version of the data, or refusing to sign a controversial message but being cooperative otherwise. Assuming all contributions are faithfully delivered, even in the pre-signing phase, may not be realistic.

To alleviate DoS attacks, *identifiable abort* (IA) is recently considered [18], [4], [7]. Honest signers can identify misbehaving parties and restart the signing process from scratch without them. The latter two results [4], [7] work under the dishonest majority setting and serve as our competitors for comparison, particularly Castagnos *et al.* [7] for the lower communication cost (than [4]). While IA seems useful, we desire more in practice. We want a "real" threshold system that minimizes DoS disturbances induced by a $(t-1)$-adversary and allows continuation as long as a threshold number of honest parties remains. We call such a robustness notion *self-healing*.

B. Technical Challenges

This paper shows that self-healing and cheater identification are achievable throughout the key generation, offline pre-signing, and online signing. A core idea is to use *distributed randomness generation* (DRG) for both the signing key and the random nonce in pre-signing. Realizing this idea securely and efficiently is a dedicated task even though secure multiparty computation (SMC) techniques are widely available. Concretely, t-out-of-n signing adds much more complexity to the prevailing n'-out-of-n' designs. At least redundancy has to be introduced, resulting in a higher communication overhead.

We aim at a minimal addition of communication rounds and costs for fault identification, such that recovering from faulty behavior and resuming the signing process will outperform restarting existing schemes from scratch. We remark that the availability of correct pre-signature shares from *all* signers, as assumed by Gągol *et al.* [18], simplifies the challenge of designing a robust online protocol. Intuitively, they provide the "ground truth" for checking and completing the online signing.

For self-healing, a naïve approach is to ask all signers to reveal their private inputs to prove their innocence when failure is reported, but it is insecure to continue the protocol. We can use zero-knowledge proof (ZKP) to ensure that all contributions are consistent. This requires tackling the technicalities for providing a faithful simulation and extraction of adversarial inputs in the security proof, particularly when cheaters are dismissed, followed by a recovery with security maintained.

C. Related Works

Threshold DSA can be traced back to Gennaro *et al.* [15]. Two decades later [14], with homomorphic encryption (HE) and independent commitment, threshold optimality ($n = t$) was achieved. Boneh *et al.* [3] proposed a tighter 4-round-signing version using level-1 HE and non-malleable commitment.

Recent works can be roughly classified into two paradigms from Lindell *et al.* [21] and Gennaro–Goldfeder [13]. The latter takes fewer communication rounds. A core technique is the *multiplicative-to-additive share conversion protocol (MtA)* which converts the product of secret values held by different signers into additive shares. Following [13], Canetti *et al.* [4] proposed schemes with IA, proactive security, and universally composable (UC) security, with MtA from Paillier additive HE. Castagnos *et al.* [6] used Castagnos–Laguillaumie encryption [8] for bandwidth efficiency, and later [7] considered IA and proactive security. Doerner *et al.* [11] use correlated oblivious transfer for MtA. Damgård *et al.* [10] considered the honest majority setting using MtA without ZKP.

Gagol *et al.* [18] followed Lindell *et al.* [21] and considered robustness in online signing, assuming everyone performs pre-signing honestly. Very recently, Abram *et al.* [1] proposed a non-robust scheme from a pseudorandom correlation generator.

In short, optimizing for communication rounds is a common goal, albeit with different tricks. Most existing works assume that sufficiently many (even all) signers honestly participated in pre-signing or in all stages to save communication rounds. However, there is no redundancy to ensure recovery.

D. Our Contributions

We propose a robust (t, n) threshold ECDSA scheme with 4-round pre-signing and "non-interactive" online signing. It achieves self-healing *without restarting* if at least t honest parties are involved; otherwise, in the face of a dishonest majority, we gracefully fall back to the best case: identifiable abort. Table I summarizes major constructions of (n, n) or (t, n) threshold ECDSA with fault identification. Our scheme is desirable when computational resources are scarce and in a decentralized setting where faults are easier to be induced. Given a pre-signature, a signature share can be produced without interacting with others, which is desirable in cold storage.

Our novelty lies in the identification mechanism and the simulation strategy of shares of two secrets' product. The latter allows us to use MtA for (t, n) secret sharing, in contrast to the prior (n, n) usage. Our identification mechanism allows the protocol to continue securely with a small overhead in communication and computation as the components from MtA remain relevant. We remark that our security proof extracts secret values via decryption instead of rewinding. This reduces the communication rounds and issues in the security proof[1].

II. Technical Overview

A. Background on Threshold ECDSA

Let $\mathbb{G} = \langle g \rangle$ be a group of prime order q and $\mathbb{F}_q$ be a finite field with q elements[2]. ECDSA $(\mathsf{Keygen}, \mathsf{Sign}, \mathsf{Verify})$ is a variant of DSA instantiated over a group of points on an elliptic curve as defined by $(\mathbb{G}, q, g)$ in the public parameter pp.

- $\mathsf{Keygen}(\mathsf{pp}) \to (\mathsf{vk}, \mathsf{sk})$: Randomly sample $x \leftarrow\!\!\$\ \mathbb{F}_q$ and set $X := g^x$, output $(\mathsf{vk}, \mathsf{sk}) := (X, x)$.
- $\mathsf{Sign}(\mathsf{sk}, \mathsf{msg}) \to \sigma$: Set $m := \mathsf{H}(\mathsf{msg})$, randomly sample[3] $k \leftarrow\!\!\$\ \mathbb{F}_q$ set $R := g^{1/k}$ and output $\sigma := (r, s := k(m+rx))$, where r is the x-projection of point R on $\mathbb{F}_q$.
- $\mathsf{Verify}(\mathsf{vk}, \sigma, \mathsf{msg}) \to 0/1$: Parse $\sigma = (r, s)$, accept if r is the x-projection of $R' := (g^m X^r)^{1/s} \in \mathbb{G}$ where $m := \mathsf{H}(\mathsf{msg})$.

In the distributed setting, k and x are both secret-shared. For $g^{1/k}$, which we call the *signature nonce*, one can use the inversion trick by Beaver with the help of another secret-shared random factor γ. We call $k\gamma$ a *"pseudo-nonce"* since it composes of the (signature) nonce. Similarly, we call kx the *"pseudo-key."* With g^γ, we can compute $g^{1/k}$ by $(g^\gamma)^{\frac{1}{k\gamma}}$.

MtA is used to derive $k\gamma$. (kx can be derived similarly.) Signers first generate random (additive) shares of (k, γ). Each party i then locally converts the (Shamir) share of the signing key x_i into additive shares by multiplying the Lagrange coefficient. To compute $k\gamma = (\sum_i k_i)(\sum_j \gamma_j) = \sum_{i,j} k_i\gamma_j$ without leaking the shares k_i and γ_j, the multiplied term $k_i\gamma_j$ is converted into additive shares via a 2-party MtA run by parties i and j. After that, each party reveals the sum of all the received shares and reconstructs the pseudo-nonce accordingly. To defend against malicious adversaries on top of MtA, Gennaro and Goldfeder [13] built an *MtA with check*

[1]Non-interactive ZKP from Fiat–Shamir heuristic may cause an exponential number of rewinding under a multiparty setting [26].

[2]The lower case letters $(x, \gamma, k, \ldots)$ are usually secret elements in $\mathbb{F}_q$, and their public values in $\mathbb{G}$ are denoted by capital letters $(X = g^x, \Gamma, K, \ldots)$.

[3]Resample if x, k, r or s is 0. Likewise, Verify should reject $(r, s) = (0, 0)$.

TABLE I: Comparison of (n, n) or (t, n) Threshold ECDSA Schemes with Fault Identification

Schemes	Communication Rounds		Protection	Cost	Proof of Security	Threshold Type
Paillier-based [4]	Offline: 3 Offline: 6	Online: 1 Online: 1	Identifiable Abort Identifiable Abort	$\mathcal{O}(n^2)$ $\mathcal{O}(n)$	UC framework UC framework	Fixed size-n group Fixed size-n group
CL-based [7]	Offline: 6	Online: 1	Identifiable Abort	$\mathcal{O}(n)$	Game-based	Fixed size-n group
SMC-based [18]	Offline:10	Online: 3	Abort Robust Online Signing	$\mathcal{O}(n)$	Simulation-based	Fixed size-n group in Pre-Sign Any size-$\geq t$ subgroup in Sign
Ours	Offline: $4(6)$	Online: 1	Self-healing & Cheater Id.	$\mathcal{O}(n)$	Game-based	Any size-$\geq t$ subgroup

If the identification does not require an additional communication round, the size of verification materials is counted as its cost. In the worst case (culprit detected in every stage), our scheme requires a 6-round pre-signing.

(MtAwc) protocol. Secret values (*e.g.*, a) are checked against public terms by "exposing" them in exponent (*e.g.*, g^a).

After knowing the hashed message m, each party i reveals its signature share $k_i m + r[kx]_i$, where $[kx]_i$ is the party i's additive share of kx. They are (n', n') shares of the desired ECDSA signature, where n' is the number of participants.

B. Three Components of Our Construction

Our construction works in a truly (t, n') setting. The number of participants n' can decrease. The first ingredient is the distributed randomness generation (DRG), inspired by robust distributed key generation (DKG) [16], followed by share revelation for multiplying threshold shares using 2-party MtA. We also propose new verification machinery to verify the product of shares from the 2-party MtA protocol for achieving self-healing (robustness) via removing problematic shares.

1) Distributed Randomness Generation: Our DRG works in the dishonest majority setting and only requires 3 communication rounds in the worst case. In Phase 1, each party i in the participating party set $\mathcal{P}$ acts as the dealer and uses Pedersen verifiable secret sharing (Pedersen VSS) to distribute shares $\{\chi_{ij}\}_j$ of an initial secret χ_i and its verification materials over authenticated channels (as in [7]). The receiver j locally verifies χ_{ij} for each i, and, if the verification fails, broadcasts it as the complaint. Other parties can verify the complaint and disqualify the problematic dealer i locally. This phase would then conclude with correct shares of χ_i distributed.

In Phase 2, each party i homomorphically combines the (initial) shares $\{\chi_{ji}\}_j$ to generate a new share $x_i = \sum_{j \in \mathcal{Q}} \chi_{ji}$ for a qualified set $\mathcal{Q}$ of all parties who passed Phase 1. The parties broadcast share in exponent g^{x_i} with ZKP of well-formedness with respect to the committed polynomial coefficients (formed by homomorphically combining the VSS materials in Phase 1). This step identifies any cheaters, which can then be kicked out such that the protocol continues with only correct shares, finally contributing to the recovery/construction of the public key g^x. With the two modifications, cheater identification in the first phase requires one round of interaction, and no interaction is needed for the second phase.

On top of the complaint mechanism, the parties also send the ciphertext of its secret and the combined share in Phases 1 and 2. The encrypted shares can be used in the MtA protocol for homomorphic evaluation over shares for offline signing. They allow simple knowledge extraction in the security proof.

2) Share Revelation: The existing usage of 2-party MtA (Section II-A) (for pseudo-nonce) outputs additive shares α_{ij}, β_{ij} of $k_i\gamma_j = \alpha_{ij} + \beta_{ij}$ to party i and j, respectively. Reconstruction of $k\gamma$ does not tolerate even a single fault since it requires all of $\{k_i\gamma_j\}_{i,j}$ and hence $\{\alpha_{ij}, \beta_{ij}\}_{i,j}$ terms.

We use DRG to generate (t, n') shares of k and γ, *i.e.*, $k = \sum_i L_i k_i$ and $\gamma = \sum_i L_i \gamma_i$ for Lagrange coefficients L_i. Reconstructing $k\gamma$ from a set of parties is tricky as the Lagrange coefficient varies with the set. Simply revealing party i's shares $(\alpha_{ij}, \beta_{ji})$ leaks the share k_i via $k_i = (\alpha_{ij} + \beta_{ij})/\gamma_j$ since party j knows (β_{ij}, γ_j), where $\alpha_{ij} + \beta_{ij} = k_i\gamma_j$.

To avoid direct revelation, party i could locally sum up the shares before releasing them [7], [4]. However, it does not work if k and γ are threshold-shared. If we multiply the shares by the Lagrange coefficient before broadcasting, *i.e.*, sending $L_i^2 k_i \gamma_i + \sum_{j \neq i} L_i L_j (\alpha_{ij} + \beta_{ij})$, they degenerate into (n', n') shares with the set of n' participants fixed.

A crucial observation we made is that the terms $k_i\gamma_j$ and $k_j\gamma_i$ are multiplied by the same Lagrange coefficient for the reconstruction equation of $k\gamma$ arranged as below:

$$\sum_i L_i k_i \sum_i L_i \gamma_i = \sum_{i,j<i} L_i L_j (k_i\gamma_j + k_j\gamma_i) + \sum_i L_i^2 k_i \gamma_i. \quad (1)$$

Hence, we can combine party i's additive shares of these two terms (*i.e.*, $\alpha_{ij} + \beta_{ji}$) without affecting the reconstruction. In this approach, we group the terms by their Lagrange coefficient, which extends the size of each party's share to n'.

For security, each party i masks the additive share $\alpha_{ij} + \beta_{ji}$ for each j by Shamir secret-shares $\{\theta_{ij}\}_{j \in [n']}$ of 0 [10], [24] and publishes the size-n' set of masked shares $\{\delta_{ij}\}_{j \in [n']} := \{k_i\gamma_i + \theta_{ii}\} \cup \{\alpha_{ij} + \beta_{ji} + \theta_{ij}\}_{j \neq i}$ (with an abused notation, while the assignment order is identified by their subscripts). The parties can then compute $k\gamma = \sum_{i,j}(L_i L_j \delta_{ij})$.

The above tackles the reconstruction issue and appears to be intuitively secure. Yet, it introduces a technical hurdle in our security proof – given a set of shares, a simulator needs to mold the remaining shares such that the reconstruction from any t-subset of shares always gives a particular result. If we shoot for an (n', n') sharing, the reconstruction is merely the sum of all shares; one can easily simulate a consistent transcript (share) by subtracting all others' shares from a particular result.

For (t, n') threshold sharing, one can simulate a share by plugging $(t-1)$ shares and the reconstruction result into the reconstruction equation. For our Equation (1), simulating shares

$\{\delta_{ij}\}_{j\in[n']}$ of a single party i (controlled by the simulator) involves at least t shares (among $\{\delta_{ij}\}_{j\in[n']}$). There are two rounds of Lagrange interpolations, *i.e.*, $k\gamma = \sum_i L_i\delta_i$ and $\delta_i = \sum_j L_j\delta_{ij}$. Picking different subsets of parties for the reconstruction equation can generate a system of equations to simulate all shares. Unfortunately, this strategy generates $\sum_{i=t}^{n}\binom{n}{i}$ equations, forming an over-determined system of equations[4] with $(n-(t-1))n$ unknowns. How to efficiently solve such an over-determined system remains unknown.

Our trick is to write the targeted result $k\gamma$ as a product of two arbitrary values a, b and then simulate with respect to their shares. More concretely, $ab = \sum_{i\in[n'],j<i} L_iL_j(a_ib_j + a_jb_i) + \sum_{i\in[n']} L_i^2 a_ib_i$, where a_i, b_j are shares of a, b respectively such that $a_ib_j + a_jb_i$ is a share of $k\gamma$. From the share δ_{ji} of the adversary, we simulate the share δ_{ij} as $a_ib_j+a_jb_i-\delta_{ji}$ such that the share $a_ib_j + a_jb_i$ is generated during reconstruction. We can then simulate each share one by one.

3) Cheater Identification: The parties need to backtrack the error source when the pre-signature (*e.g.*, the pseudo-nonce) or the final signature is invalid. Existing IA schemes [4], [7] just reveal the constitutes (including the shares of randomness k, γ, and the MtA additive shares) of all pre-signatures. However, leaking randomness mandates an abort.

Another approach is to prove the well-formedness of the ciphertexts (used by the MtA protocol) by ZKP. This approach is not ideal in the Paillier-based regime, incurring $\mathcal{O}(n^2)$ communication cost for identification that works over a fixed size-n group [4], albeit the offline signing only takes 3 rounds. The cost comes from a verifier-specific ZKP for showing a correct operation in the MtA protocol. Each pair of parties needs to prove every message between them individually.

We propose a cheater identification machinery verifying the (reconstructed) pseudo-nonce and signature without leaking any secret share (*e.g.*, from DRG). Its non-triviality lies in the "unstructured" threshold shares $\{\delta_{ij}\}_j$ due to the conversion during MtA and randomization before share revelation. Our trick is to use additive shares in exponent $\{g^{\beta_{ij}}\}_j$ broadcasted by party i during MtAwc to verify $\{\delta_{ij}\}_j$ (*cf.*, SectionV-B3) against $g^{k_i\gamma}$. Notice that $\sum_j L_j\delta_{ij} + \sum_j L_j(\beta_{ij} - \beta_{ji}) = L_ik_i\gamma_i + \sum_{j\neq i} L_j(\alpha_{ij} + \beta_{ji} + \beta_{ij} - \beta_{ji}) = L_ik_i\gamma_i + \sum_{j\neq i} L_j(\alpha_{ij}+\beta_{ij}) = L_ik_i\gamma_i + \sum_{j\neq i} L_j(k_i\gamma_j) = k_i\gamma$. Lifting it to exponent allows us to utilize[5] $g^{\beta_{ji}}, g^{\beta_{ij}}$ to compute $g^{k_i\gamma}$. As this partial reconstruction only involves contribution from party i, any fault on the computed $g^{k_i\gamma}$ is imputed to this party.

C. (In)extensibility to Security against Adaptive Corruption

Achieving (t, n) threshold signing also makes adaptive security more challenging. The Paillier-based work with UC security [4] provides an adaptively-secure n-out-of-n construction and refers to the base work [13] for a t-out-of-n extension, but without specifying whether the extension is still adaptively-secure. Meanwhile, the CL-based work [6] provides an adaptively-secure construction for $t = n-1$ by the following strategy, which explains the (in)extensibility to adaptive security. In short, one "special player"/signer helps the game-based simulation. If this signer is corrupted, the simulation fails, and hence the simulator rewinds to make this signer uncorrupted throughout the simulation. The probability of success is $1/n$. For t-out-of-n, the probability becomes $1/\binom{n}{t}$.

III. Preliminary

A. Online/Offline (t, n) Threshold ECDSA

Let $\mathcal{P}$ be the participant set. When each party $i \in [1, n]$ inputs in_i to a probabilistic polynomial-time (PPT) protocol P and obtains out_i, it is denoted by $\mathsf{P}\langle \mathsf{in}_1; \ldots; \mathsf{in}_n\rangle \to \langle \mathsf{out}_1; \ldots; \mathsf{out}_n\rangle$. $\mathsf{Alg}(\mathsf{in}_i) \to \mathsf{out}_i$ denotes party i running Alg.

Assuming the use of public key encryption, a (t, n) threshold ECDSA scheme features an ideally decentralized setup that outputs pp when given a security parameter λ, a threshold key generation protocol $\mathsf{TKeygen}$, and a threshold signing protocol. $\mathsf{TKeygen}\langle \mathsf{pp}; \ldots; \mathsf{pp}\rangle \to \langle(\mathsf{vk}, \mathsf{pk}, \mathsf{sk}_1); \ldots; (\mathsf{vk}, \mathsf{pk}, \mathsf{sk}_n)\rangle$ outputs to each party i an ECDSA verification key vk, a public key pk (which contains a set of public encryption keys of all parties), and a threshold signing key sk_i (including party i's threshold ECDSA key share x_i and decryption key).

The threshold signing protocol outputs a signature σ on an agreed message msg if a subset of $n' \geq t$ signers participate honestly with their threshold signing keys $\{\mathsf{sk}_i\}_{i\in[n']}$, assuming w.l.o.g. the first n' signers form the subset. In the preprocessing model, the threshold signing protocol is split into the (threshold) offline pre-signing protocol and threshold online signing protocol ($\mathsf{PreSign}, \mathsf{TSign}$):

- $\mathsf{PreSign}\langle(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_1); \ldots; (\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_{n'})\rangle \to \langle \psi_1; \ldots; \psi_{n'}\rangle$
- $\mathsf{TSign}\langle(\mathsf{pp}, \mathsf{pk}, \mathsf{vk}, \mathsf{msg}, \mathsf{sk}_1, \psi_1); \ldots)\rangle \to \sigma$

The online signing protocol uses the (message-independent) pre-signature (shares) $\{\psi_i\}$ from offline $\mathsf{PreSign}$ to generate a signature σ. Looking ahead, our formulation of ψ_i consists of the actual pre-signature φ_i and its verification materials V_i.

B. Security Models and Communication Channels

Following prior works [4], [7] with IA, we use authenticated and synchronized broadcast channels. Synchronized broadcast ensures bounded communication delay to hold irresponsive signers accountable. Private messages (*e.g.*, secret shares) can be encrypted before broadcasting. We consider a PPT adversary $\mathcal{A}$ that corrupts at most $(t-1)$ parties at the very beginning. It can ask any corrupted party to deviate from the protocol specification at any time (malicious adversary) and submit the message after viewing others (rushing adversary).

Definition 3.1 (Enhanced Existential Unforgeability [4]): Consider $\mathcal{A}$ that is PPT and given the public output (with vk) of $\mathsf{TKeygen}(\mathsf{pp})$ and adaptive accesses to the oracles below:

- $\mathsf{O}^{\mathsf{Rand}}$: Return a uniformly random point $R = (r_x, r_y) \in \mathbb{G}$;
- $\mathsf{O}^{\mathsf{Sign}}(\mathsf{sk}, \mathsf{msg}; R)$: On input nonce R and message msg, return signature (r, s) on msg with $r := r_x \bmod q$ if R is generated by $\mathsf{O}^{\mathsf{Rand}}$ and is never used; return $\perp$ otherwise.

Let $\mathcal{Q}$ be the set of queried messages. ECDSA is existentially unforgeable under chosen message attack if the probability that $\mathcal{A}$ outputs a valid signature σ^* on $\mathsf{msg}^* \notin \mathcal{Q}$ is negligible in λ.

[4]For a participant set $\mathcal{P}$ of size n', the simulator controls $\mathcal{H}$ (size $n' - t + 1$) and simulates $n'(n' - t + 1)$ shares in total. Robustness requires $\sum_{i\in\mathcal{P}'} L_{i,\mathcal{P}'} \sum_{j\in\mathcal{P}'} L_{j,\mathcal{P}'}\delta_{ij} = \delta$ to hold for any $\mathcal{P}' \subseteq \mathcal{P}$. The simulator computes the shares $\{\delta_{ij}\}_{i\in\mathcal{H},j\in\mathcal{P}}$ with respect to the above equation.

[5]$\{\delta_{ij}\} = \{k_i\gamma_i + [0]_i\} \cup \{\alpha_{ij} + \beta_{ji} + [0]_j\}_{j\neq i}$, with $[0]_i$ as a share of 0.

Groth and Shoup [19] show that adaptively choosing messages after knowing nonces enables a new attack, motivating the ratio-resistance assumption on the function H[6].

Definition 3.2 (Threshold Signature Unforgeability [7]): An online/offline threshold signature scheme is unforgeable if, for any PPT adversary $\mathcal{A}$ that corrupted at most $t-1$ parties and is given the views of $\mathsf{TKeygen}$ and $\mathsf{PreSign}$, and the output of TSign on inputs of messages from an adaptively chosen set $\mathcal{Q}$ and outputs of $\mathsf{PreSign}$, the probability that $\mathcal{A}$ outputs a valid signature σ^* for $m^* \notin \mathcal{Q}$ is negligible in λ.

Protocol continuation depends on the majority setting. Under the dishonest majority setting, only identifiable abort can be achieved, whereas the honest parties can identify corrupted parties when the protocol aborts. If there are a sufficient number t of honest parties (*i.e.*, an honest majority), robustness could be achieved, which ensures the protocols, including $\mathsf{TKeygen}$, $\mathsf{PreSign}$, and TSign, will complete successfully.

At a high level, a (t, n) threshold ECDSA scheme is robust if for n participating parties $\mathcal{P} = \{1, \ldots, n\}$ and any set of corrupted parties $\mathcal{C} \subset \mathcal{P}$ of size $t-1$, and for keys obtained from $\mathsf{TKeygen}$ run by $\mathcal{P}$, it holds that $\mathsf{PreSign}$ returns the pre-signature such that the remaining honest parties can output a valid ECDSA signature on any message m (via TSign) under the public key from $\mathsf{TKeygen}$.

C. Verifiable Secret Sharing

Verifiable secret sharing VSS allows share verification. Pedersen VSS [23] uses two instances of Shamir secret sharing SS over polynomials $f(z), f'(z)$. Pedersen commitments of the coefficients of $f(z)$, with $f(0)$ being the secret to share, are given using coefficients of $f'(z)$ as the randomness. This set of commitments serves as a committed polynomial. One can verify the received shares by evaluating the committed polynomial via the homomorphic property of Pedersen commitments. We recall the algorithms[7] of Pedersen VSS for the set of parties $\mathcal{P}$.

- $\mathsf{VSS.Share}(\chi, \mathcal{P}) \to (\{\chi_j, \chi'_j\}_{j\in\mathcal{P}}, F)$:
  1) Sample $a_d, a'_d \leftarrow\!\!\$\ \mathbb{F}_q$, $\forall d \in [0, t-1]$. Reassign $a_0 := \chi$.
  2) Define $f(z) := \sum_{d=0}^{t-1} a_d z^d$ and $f'(z) := \sum_{d=0}^{t-1} a'_d z^d$.
  3) Set $F := \{F_d\}_{d\in[0,t-1]} = \{g^{a_d} h^{a'_d}\}_{d\in[0,t-1]}$.
  4) Output $\{\chi_j, \chi'_j\}_{j\in\mathcal{P}} := \{f(j), f'(j)\}_{j\in\mathcal{P}}$ and F.
- $\mathsf{VSS.Verify}((\chi_j, \chi'_j), F)$: Parse $F = \{F_d\}_{d\in[0,t-1]}$. Output 1 if $\prod_{d=0}^{t-1} F_d^{j^d} = g^{\chi_j} h^{\chi'_j}$; 0 otherwise.

Lemma 1 ([17]): Pedersen VSS satisfies the following properties under the discrete logarithm assumption.

- Correctness: Given any subset $\mathcal{P}' \subseteq \mathcal{P}$ with more than t shares $\{\chi_i, \chi'_i\}_{i\in\mathcal{P}'}$, all of them passed $\mathsf{VSS.Verify}$, the reconstruction algorithm uniquely reconstructs the secret χ.
- Robustness: In the presence of malicious parties, if there are more than t honest parties (shares that pass Verify), the reconstruction algorithm always reconstructs the secret χ.
- Unconditional secrecy: The view of any adversary $\mathcal{A}$, who corrupts up to $(t-1)$ parties, is independent of the secret χ.

[6]Informally, it is hard to find two values such that the ratio of the hashes of these two values satisfies the challenger-chosen value; see [19, Theorem 3].

[7]We omit the simple definition of $\mathsf{SS.Share}$, and the reconstruction algorithm (a simple polynomial interpolation) because our scheme never calls it.

User $u_a(g^b, a, (\mathsf{dk}_{u_a}))$		**User** $u_b(g^b, b)$
$c_a \leftarrow \mathsf{Enc}(\mathsf{ek}_{u_a}, a)$	$\xrightarrow{c_a}$	$\beta \leftarrow \mathbb{Z}_q$
$\alpha \leftarrow \mathsf{Dec}(\mathsf{dk}_{u_a}, c_\alpha)$	$\xleftarrow{c_\alpha, g^\beta}$	$c_\alpha := b \otimes c_a \oplus \mathsf{Enc}(\mathsf{ek}_{u_a}, -\beta)$
check $g^\alpha \cdot g^\beta \stackrel{?}{=} (g^b)^a$		

Fig. 1: Mult.-to-Add. Share Conversion with Check (MtAwc)

D. Class-Group-based Encryption

Our scheme employs linearly-homomorphic Castagnos–Laguillaumie (CL) encryption [8] that works in the composite-order group $\mathfrak{F} \times \mathfrak{G}^q$, where $\mathfrak{F}$ is a group of known order q generated by $\mathfrak{f}$ and $\mathfrak{G}^q$ is an unknown order-s group generated by $\mathfrak{g}_q$. A message $m \in \mathbb{F}_q$ is encoded as $\mathfrak{f}^m \in \mathfrak{F}$, where the discrete logarithm problem is easy. Its prime order q frees us from range proof needed (*e.g.*, in MtAwc) by other HE schemes. It is secure under the hard subgroup membership assumption ($\mathfrak{G}^q$ vs. $\mathfrak{F} \times \mathfrak{G}^q$). We assume $\mathcal{D}_q = [0, 2^{\lambda_d} q\hat{s}]$, where $\hat{s}$ is a bound of s provided by the class group.

Definition 3.3: CL encryption (CLE) is defined by:

- $\mathsf{KGen}(\mathsf{pp}) \to (\mathsf{ek}, \mathsf{dk})$: Output $\mathsf{dk} \leftarrow\!\!\$\ \mathcal{D}_q$ and $\mathsf{ek} := \mathfrak{g}_q^{\mathsf{dk}}$.
- $\mathsf{Enc}(\mathsf{ek}, m; r) \to c_m$: Pick $r \leftarrow\!\!\$\ \mathcal{D}_q$, output $(\mathfrak{g}_q^r, \mathfrak{f}^m \mathsf{ek}^r)$.
- $\mathsf{Dec}(\mathsf{dk}, (c_1, c_2)) \to m$: Output the discrete logarithm of c_2/c_1^{dk} to the base $\mathfrak{f}$.
- $\mathsf{Eval}(c_m, c_{m'}, +) \to c_{m+m'}$: Parse $c_m = (c_1, c_2)$ and $c_{m'} = (c'_1, c'_2)$, output $(c_1 \cdot c'_1, c_2 \cdot c'_2)$, denoted by $c_m \oplus c_{m'}$.
- $\mathsf{Eval}(a, c_m, \times) \to c_{am}$: Parse $c_m = (c_1, c_2)$, output $c_{am} := (ac_1, ac_2)$, also denoted by $a \otimes c_m$.

E. Multiplicative-to-Additive Share Conversion (with Check)

MtA converts the multiplicative shares ab into additive shares $\alpha + \beta$ securely, where (a, α) and (b, β) are owned by two different parties. Assume that u_a is the receiver who publishes its public key of a linearly-homomorphic encryption scheme[8], and the sender u_b publishes its secret in exponent g^b. We employ the MtA with check (MtAwc) protocol [13] $\mathsf{MtAwc}\langle(g^b, a, (\mathsf{ek}_{u_a}, \mathsf{dk}_{u_a})); (g^b, b, \mathsf{ek}_{u_a})\rangle \to \langle(g^\beta, \alpha); (\beta)\rangle$ in Figure 1. The additive share in exponent g^β allows homomorphic operations for checking and verification in later steps.

F. Zero-Knowledge Proof

Let x be a statement of an NP language $\mathcal{L}$ and w be its witness. A non-interactive zero-knowledge (NIZK) proof NIZK allows a prover to convince the verifier that (x, w) is in relation $\mathcal{R}$ without revealing w. With Fiat–Shamir heuristic, a NIZK proof system $(\mathsf{Prove}, \mathsf{Verify})$ over the standard Σ-protocol can be built using a hash function $\mathsf{H}_{\mathsf{NIZK}} : \{0,1\}^* \to \mathbb{C}$, where $\mathbb{C}$ is the challenge space depending on the relation, *e.g.*, $\mathbb{F}_q$.

- Via $\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Prove}(x; w) \to \pi$, the prover generates proof π with respect to the committed message A.
- Via $\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Verify}(x, \pi) \to 0/1$, anyone can verify proof π.

[8]In our case, ciphertexts of a, b are made public before running MtAwc.

Our scheme needs NIZK for the below NP-relations. We start with the underlying Σ-protocol, which we review the definition in Appendix A-A and instantiations in Appendix A-B.

1) Committed Exponent: For a commitment PC and a group element Q, the following relation verifies that m committed in PC is the exponent of Q to the base element g_0.

$$\mathcal{R}_{\mathsf{DL\text{-}PC}} = \{((\mathsf{PC}, g_0, Q), (m, r)) : Q = {g_0}^m \wedge \mathsf{PC} = g^m h^r\}$$

2) Equality of Committed Values and One Randomness: For commitments $\mathsf{PC}, \mathcal{N}$, and a group element $\mathcal{M}$, the following relation verifies that k is committed in both PC and $\mathcal{N}$ with bases (g, h) and $(X, 1/g)$, respectively, and μ is the exponent of $\mathcal{M}$ to the base R and also the commitment opening of $\mathcal{N}$.

$$\mathcal{R}_{\mathsf{DL\text{-}2PC}} = \{((\mathsf{PC}, \mathcal{N}, \mathcal{M}, X, R), (k, k', \mu)) : \\ \mathsf{PC} = g^k h^{k'} \wedge g^{\mu}\mathcal{N} = X^k \wedge \mathcal{M} = R^{\mu}\}$$

3) CL Key Pair: $\mathcal{R}_{\mathsf{key}} = \{(\mathsf{ek}, \mathsf{dk}) : \mathsf{dk} \in [0, S] \wedge \mathsf{ek} = \mathfrak{g}_q^{\mathsf{dk}}\}$

4) CL Encryption of a Committed Value:

$$\mathcal{R}_{\mathsf{Enc\text{-}PC}} = \{((\mathsf{PC}, \mathsf{ek}, c), (m, r, \rho)) : \\ \rho \in [0, S] \wedge c = \mathsf{Enc}(\mathsf{ek}, m; \rho) \wedge \mathsf{PC} = g^m h^r\}$$

5) CL Decryption: The proof is simplified by revealing m from decrypting (c_1, c_2), and proving about $c_2' = c_2/\mathfrak{f}^m$.

$$\mathcal{R}_{\mathsf{Dec}} = \{((\mathsf{ek}, (c_1, c_2')), \mathsf{dk}) : \mathsf{dk} \in [0, S] \wedge c_2' = c_1^{\mathsf{dk}} \wedge \mathsf{ek} = \mathfrak{g}_q^{\mathsf{dk}}\}$$

IV. Distributed Randomness Generation

To support self-healing in multiparty protocols, we define a distributed randomness generation DRG protocol, generalizing the existing DKG protocol for different forms of randomness, *e.g.*, committed shares for verification or encrypted shares for further computation. We use DRG as a major building block.

A. Definition

$\mathsf{DRG} = (\mathsf{Gen}, \mathsf{GenVf}, \mathsf{Comb}, \mathsf{CombVf}, \mathsf{RevealExp}, \mathsf{ExpVf})$ aims to generate shares $\{x_j\}_j$ of a secret x with "public" shares $\{\mathsf{pub}_{x_j}\}$ for verification against a certain relation $\mathcal{R}$. For example, pub_{x_j} may contain commitment or encryption of x_j. RevealExp and ExpVf are optional, which reveal shares in exponent X_i of shares x_i verifiably, *e.g.*, $X_i := g^{x_i}$ for Pedersen VSS share (x_i, x_i'). The shares in exponent $\{X_j\}_j$ allow reconstruction of the secret in exponent $g^x = \prod_j X_j^{L_j}$, which is useful when g^x serves as a public key with shares $\{x_j\}_j$ of the secret key x distributed. We define (Pedersen VSS-based) DRG as follows, assuming they are run by party i.

- $\mathsf{Gen}(\mathsf{pp}) \to (\{\chi_{ij}, \chi'_{ij}\}_j, C_{\chi_i}), \pi_{\chi_i}$: On input the public parameter[9] pp, Gen outputs initial secret shares $\{\chi_{ij}, \chi'_{ij}\}_j$, verifying materials C_{χ_i} of initial secret χ_i, and proof π_{χ_i}.
- $\mathsf{GenVf}(\chi_{ij}, \chi'_{ij}, C_{\chi_i}, \pi_{\chi_i}) \to 0/1$: On input the shares (χ_{ij}, χ'_{ij}), verifying material C_{χ_i}, and proof π_{χ_i}, GenVf outputs a bit indicating the validity of inputs.
- $\mathsf{Comb}(\{\chi_{ji}, \chi'_{ji}\}_j) \to (w_i, \mathsf{pub}_{x_i}, \pi_{x_i})$: On input the shares $\{\chi_{ji}, \chi'_{ji}\}_j$ of the secrets $\{\chi_j\}_j$, Comb outputs private output w_i (containing x_i), public share pub_{x_i}, and proof π_{x_i}.
- $\mathsf{CombVf}(\{C_{\chi_j}\}_j, \mathsf{pub}_{x_i}, \pi_{x_i}) \to 0/1$: On input the set of verifying materials $\{C_{\chi_j}\}_j$, the public share pub_{x_i}, and proof π_{x_i}, it outputs a bit indicating the validity of inputs.
- $\mathsf{RevealExp}(w_i, \mathsf{pub}_{x_i}) \to (X_i, \pi_{X_i})$ On input the private output w_i (containing x_i) and the public share pub_{x_i}, it outputs the private share in exponent X_i and proof π_{X_i}.
- $\mathsf{ExpVf}(\mathsf{pub}_{x_i}, X_i, \pi_{X_i}) \to 0/1$: It verifies the public share pub_{x_i}, the share in exponent X_i, and proof π_{X_i}.

DRG starts by having every party run Gen(pp). They then send the resulting initial shares $\{\chi_{ij}, \chi'_{ij}\}_j$ to each party j and broadcast the public outputs (proof and verification materials). Next, all parties j run $\mathsf{Comb}(\{\chi_{ji}, \chi'_{ji}\}_j)$ on all shares verified by GenVf to output secret share w_j. DRG concludes after CombVf verified all outputs so far. DRG is correct if:

1) x defined by $\{w_i\}$ output by Comb is uniformly distributed.
2) All subsets of secret output w_i provided by more than t honest parties define the same established secret x.
3) All subsets of public-secret output pairs $(\mathsf{pub}_{x_i}, w_i)$ provided by more than t honest players satisfy the relation $\mathcal{R}$. Also, X_i is (one of) the secret output w_i in exponent.

We say DRG is self-healing if the protocol involves at least t honest parties, the protocol always completes successfully. Secrecy requires that, for any PPT adversary $\mathcal{A}$ having corrupted at most $t-1$ parties, there exists a simulator on input $X = g^x$ simulating an indistinguishable view of the protocol for $\mathcal{A}$ that outputs (pub, X), *i.e.*, $\mathcal{A}$ learns nothing about x, except pub, where $\mathsf{pub} = \{\mathsf{pub}_{x_j}\}$ is the set of all parties' public output.

B. Distributed Randomness Generation Construction

Figure 2 presents our DRG protocol with the generation, combination, and output phases. For brevity, all algorithms implicitly take in encryption keys $\{\mathsf{ek}_j\}$ of all parties.

The protocol is run by an *updating set of parties* $\mathcal{P}$ that keeps removing identified cheaters. At the end of the generation phase, the distributively-generated secret is fixed by the contribution from the honest parties. We call them the *qualified set* $\mathcal{Q}_x$, which is a fixed set that defined a fixed secret x. $\mathcal{P}$ may update to remove cheaters detected in GenVf, and $\mathcal{Q}_x := \mathcal{P}$ is fixed afterward. CombVf may remove more cheaters from $\mathcal{P}$ but will not affect $\mathcal{Q}_x$.

1) Generation phase ($\mathsf{Gen} \leftrightarrow \mathsf{GenVf}$*):* Assume $|\mathcal{P}| = n$; we aim to distribute (t, n) verifiable shares of initial secret χ_i sampled by party $i \in \mathcal{P}$ to all other parties in this phase.

(Gen.) Party i randomly picks and encrypts[10] its initial secret χ_i under its public key ek_i. Party i (t, n)-shares χ_i using VSS.Share, which outputs two sets of shares $\{\chi_{ij}, \chi'_{ij}\}_{j \in \mathcal{P}}$ and committed polynomial coefficients $\{F_{\chi_i,d}\}_{d \in [0,t-1]}$ (*cf.*, Section III-C). The share (χ_{ij}, χ'_{ij}) is sent to party $j \in \mathcal{P} \setminus \{i\}$ via a private authenticated channel. Party i sets and broadcasts the verification material $C_{\chi_i} = (c_{\chi_i}, \{F_{\chi_i,d}\}_{d \in [0,t-1]})$ and proof π_{χ_i} for Enc-PC that c_{χ_i} encrypts χ_i committed in $F_{\chi_i,0}$ as the evaluation of the committed polynomial at 0.

(GenVf.) Party j verifies shares (χ_{ij}, χ'_{ij}) from party i by running $\mathsf{VSS.Verify}((\chi_{ij}, \chi'_{ij}), F_{\chi_i})$ and checking proof π_{χ_i}.

[9]Our construction puts public encryption keys $\{\mathsf{ek}_i\}$ of participants in pp.

[10]It allows extracting secret χ_i using decryption key dk_i without rewinding.

```
DRG Algorithms

$\mathsf{Gen}(\mathsf{pp}) \to (\{\chi_{ij}, \chi'_{ij}\}_{j\in\mathcal{P}}, C_{\chi_i}, \pi_{\chi_i})$
1 : $\chi_i \leftarrow\$ \mathbb{F}_q, c_{\chi_i} \leftarrow \mathsf{Enc}(\mathsf{ek}_i, \chi_i; \rho_{\chi_i})$
2 : $(\{\chi_{ij}, \chi'_{ij}\}_{j\in\mathcal{P}}, F_{\chi_i}) \leftarrow \mathsf{VSS.Share}(\chi_i, \mathcal{P})$
3 : $\textbf{parse } F_{\chi_i} \textbf{ as } \{F_{\chi_i,d}\}_{d\in[0,t-1]}$
4 : $\pi_{\chi_i} \leftarrow \mathsf{NIZK}_{\mathsf{Enc\text{-}PC}}.\mathsf{Prove}((F_{\chi_i,0}, \mathsf{ek}_i, c_{\chi_i}); (\chi_i, \chi'_i, \rho_{\chi_i}))$
5 : $\textbf{return } (\{\chi_{ij}, \chi'_{ij}\}_{j\in\mathcal{P}}, C_{\chi_i} := (c_{\chi_i}, F_{\chi_i}), \pi_{\chi_i})$

$\mathsf{GenVf}(\chi_{ij}, \chi'_{ij}, C_{\chi_i}, \pi_{\chi_i}) \to 0/1$
1 : $\textbf{parse } F_{\chi_i} \textbf{ as } \{F_{\chi_i,d}\}_{d\in[0,t-1]},\ C_{\chi_i} \textbf{ as } (c_{\chi_i}, F_{\chi_i})$
2 : $b \leftarrow \mathsf{NIZK}_{\mathsf{Enc\text{-}PC}}.\mathsf{Verify}((F_{\chi_i,0}, \mathsf{ek}_i, c_{\chi_i}), \pi_{\chi_i})$
3 : $\textbf{if } b = 1 \land \mathsf{VSS.Verify}((\chi_{ij}, \chi'_{ij}), F_{\chi_i}) \textbf{ return } 1$
4 : $\textbf{else return } 0$

$\mathsf{Comb}(\{\chi_{ji}, \chi'_{ji}\}_{j\in\mathcal{Q}_x}) \to (w_i, \mathsf{pub}_{x_i}, \pi_{x_i})$
1 : $x_i := \sum_{j\in\mathcal{Q}_x} \chi_{ji},\ x'_i := \sum_{j\in\mathcal{Q}_x} \chi'_{ji}$
2 : $\mathsf{PC}_{x_i} := g^{x_i} h^{x'_i},\ c_{x_i} \leftarrow \mathsf{Enc}(\mathsf{ek}_i, x_i; \rho_{x_i})$
3 : $w_i := (x_i, x'_i, \rho_{x_i});\ \mathsf{pub}_{x_i} := (\mathsf{PC}_{x_i}, \mathsf{ek}_i, c_{x_i})$
4 : $\pi_{x_i} \leftarrow \mathsf{NIZK}_{\mathsf{Enc\text{-}PC}}.\mathsf{Prove}(\mathsf{pub}_{x_i}; w_i)$
5 : $\textbf{return } (w_i, \mathsf{pub}_{x_i}, \pi_{x_i})$

$\mathsf{CombVf}(\{C_{\chi_j}\}_{j\in\mathcal{Q}_x}, \mathsf{pub}_{x_i}, \pi_{x_i}) \to 0/1$
1 : $\textbf{parse } \{C_{\chi_j}\}_{j\in\mathcal{P}} \textbf{ as } \{c_{\chi_j}, \{F_{\chi_j,d}\}_{d\in[0,t-1]}\}_{j\in\mathcal{P}}$
2 : $F_x := \left\{F_{x,d} := \prod_{j\in\mathcal{P}} F_{\chi_j,d}\right\}_{d\in[0,t-1]}$
3 : $\textbf{parse } \mathsf{pub}_{x_i} \textbf{ as } (\mathsf{PC}_{x_i}, \mathsf{ek}_i, c_{x_i})$
4 : $\textbf{if } \mathsf{PC}_{x_i} = \prod_{d=0}^{t-1} F_{x,d}{}^{i^d}$
5 :     $\textbf{return } \mathsf{NIZK}_{\mathsf{Enc\text{-}PC}}.\mathsf{Verify}(\mathsf{pub}_{x_i}, \pi_{x_i})$
6 : $\textbf{else return } 0$

$\mathsf{RevealExp}(w_i, \mathsf{pub}_{x_i}) \to (X_i, \pi_{X_i})$
1 : $\textbf{parse } w_i \textbf{ as } (x_i, x'_i, \rho_{x_i}),\ \mathsf{pub}_{x_i} \textbf{ as } (\mathsf{PC}_{x_i}, \mathsf{ek}_i, c_{x_i})$
2 : $\pi_{X_i} \leftarrow \mathsf{NIZK}_{\mathsf{DL\text{-}PC}}.\mathsf{Prove}((\mathsf{PC}_{x_i}, g, g^{x_i}); (x_i, x'_i))$
3 : $\textbf{return } (X_i := g^{x_i}, \pi_{X_i})$

$\mathsf{ExpVf}(\mathsf{pub}_{x_i}, X_i, \pi_{X_i}) \to 0/1$
1 : $\textbf{parse } \mathsf{pub}_{x_i} \textbf{ as } (\mathsf{PC}_{x_i}, \mathsf{ek}_i, c_{x_i})$
2 : $\textbf{return } \mathsf{NIZK}_{\mathsf{DL\text{-}PC}}.\mathsf{Verify}((\mathsf{PC}_{x_i}, g, X_i), \pi_{X_i})$
```

Fig. 2: (Threshold) Distributed Randomness Generation

If it fails, party j broadcasts the received share (χ_{ij}, χ'_{ij}) from party i as the complaint. All parties exclude i from $\mathcal{P}$[11] as π_{χ_i} is publicly verifiable. We can safely discard any cheater contribution χ_i because the combined secret x is not fixed yet. At the end of this phase, the combined secret x is fixed for the qualified set $\mathcal{Q}_x := \mathcal{P}$, *i.e.*, $x := \sum_{j\in\mathcal{Q}_x} \chi_j$. Notice that $\{\chi_j\}_{j\in\mathcal{Q}_x}$ have been correctly shared via (t, n) secret sharing.

2) Combination phase ($\mathsf{Comb} \leftrightarrow \mathsf{CombVf}$*):* We aim to generate (t, n) secret shares $\{x_i\}_{i\in\mathcal{P}}$ of the final secret $x = \sum_{j\in\mathcal{Q}_x} \chi_j$ via the linear homomorphism of VSS (*cf.*, Section III-C). This phase also outputs committed shares $\{\mathsf{PC}_{x_i} = g^{x_i} h^{x'_i}\}_{i\in\mathcal{P}}$ and encrypted shares $\{c_{x_i} = \mathsf{Enc}(\mathsf{ek}_i, x_i)\}_{i\in\mathcal{P}}$.

(Comb.) Party i homomorphically combines $\{\chi_{ji}\}_{j\in\mathcal{Q}_x}$ to generate a share $(x_i, x'_i) = (\sum_{j\in\mathcal{Q}_x} \chi_{ji}, \sum_{j\in\mathcal{Q}_x} \chi'_{ji})$ of x. It then computes a Pedersen commitment $\mathsf{PC}_{x_i} = g^{x_i} h^{x'_i}$ and an encrypted share $c_{x_i} = \mathsf{Enc}(\mathsf{ek}_i, x_i; \rho_{x_i})$ using randomness ρ_{x_i}. The public share $\mathsf{pub}_{x_i} = (\mathsf{PC}_{x_i}, \mathsf{ek}_i, c_{x_i})$ is broadcasted with proof π_{x_i} (for $\mathcal{R}_{\mathsf{Enc\text{-}PC}}$) that x_i is committed and encrypted.

(CombVf.) Upon receiving the public output pub_{x_i} and ZKP π_{x_i} from party i, party j computes the commitment PC_{x_i} of the share using the VSS verification materials $\{F_{\chi_j}\}_{j\in\mathcal{Q}_x}$ (sent after Gen and verified by GenVf). Party j verifies ZKP π_{x_i} against PC_{x_i}. If it fails, all parties set $\mathcal{P} := \mathcal{P} \setminus \{i\}$ to exclude party i, as all the inputs for verification are public.

3) Revealing secret in exponent ($\mathsf{RevealExp} \leftrightarrow \mathsf{ExpVf}$*):* (RevealExp.) After the combination phase, party i runs RevealExp to generate g^{x_i} and proof π_{X_i} for $\mathcal{R}_{\mathsf{DL\text{-}PC}}$ that it is committed inPC_{x_i}. Both will be broadcasted.

(ExpVf.) If proof π_{X_i} fails, everyone excludes i from $\mathcal{P}$.

4) Output: DRG returns the collected $\{\mathsf{pub}_{x_j}\}_{j\in\mathcal{P}}$ as public output; the combined share (x_i, x'_i) and the randomness ρ_{x_i} as the party's secret output. The public share pub_{x_i} includes ciphertext c_{x_i} encrypting x_i using randomness ρ_{x_i}, encryption key ek_i, and Pedersen commitment PC_{x_i} using opening x'_i. For $w_i := (x_i, x'_i, \rho_{x_i})$ and the public-secret key pair $(X := g^x, x)$, we require $X_i = g^{x_i}$ and $(w_i, \mathsf{pub}_{x_i}) \in \mathcal{R}_{\mathsf{Enc\text{-}PC}}$. If (RevealExp, ExpVf) are called such that $\{X_j\}_{j\in\mathcal{P}}$ are given, Any one can verifiably reconstruct $X = \prod_{j\in\mathcal{P}} X_j^{L_{j,\mathcal{P}}}$.

Appendix B will show the correctness, self-healing property, and secrecy of our DRG construction.

C. Key Refreshment

Our construction can be extended to support key refreshment as below. Notice that we have $\sum_{i\in\mathcal{U}} L_{i,\mathcal{U}} x_i = x$, the key (share) x_i now takes the role of the random secret χ_i.

(Gen $\leftrightarrow$ GenVf.) Party i first distributes shares $\{\omega_{ij}\}_j$ of x_i, *i.e.*, $x_i = \sum_{j\in\mathcal{U}} L_{j,\mathcal{U}} \omega_{ij}$.

(Comb $\leftrightarrow$ CombVf.) With respect to the change of reconstruction equation for x from $\{x_i\}_{i\in\mathcal{U}}$ (interpolation versus summation of $\{\chi_i\}_{i\in\mathcal{U}}$), we change the secret output to $w_i := (\omega_i := \sum_{j\in\mathcal{Q}} L_{j,\mathcal{Q}} \omega_{ji}, \omega'_i := \sum_{j\in\mathcal{Q}} L_{j,\mathcal{Q}} \omega'_{ji}, \rho_{\omega_i})$. Changes are also made to the committed polynomial F, and $\mathcal{Q}$ is fixed (after removing cheaters) after the generation phase.

By linearity, for any set $\mathcal{U}' \subseteq \mathcal{U}$ with $|\mathcal{U}'| \geq t$, $\sum_{i\in\mathcal{U}'} L_{i,\mathcal{U}'} \omega_i = \sum_{i\in\mathcal{U}'} L_{i,\mathcal{U}'} (\sum_{j\in\mathcal{Q}} L_{j,\mathcal{Q}} \omega_{ji}) = \sum_{j\in\mathcal{Q}} L_{j,\mathcal{Q}} (\sum_{i\in\mathcal{U}'} L_{i,\mathcal{U}'} \omega_{ji}) = \sum_{j\in\mathcal{Q}} L_{j,\mathcal{Q}} x_j = x$, so x can be correctly reconstructed from the refreshed shares ω_i.

V. Our Proposed Construction

For DRG, we first set up public parameters of Pedersen VSS, including $h \in \mathbb{G}$, by a commit-then-reveal procedure[12], and of CL encryption, which features a decentralized setup [6]. We denote the respective set of parties invoking the threshold

[11]Private shares are encrypted before transmitting via the broadcast channel, so encrypted shares from party i are either received by all or none (if party i is absent). Slandering can be prevented by proving decryption correctness.

[12]Each party first broadcasts a hash of random h_i, then broadcasts h_i in the second round and sets $h = \prod_i h_i$ without exposing its exponent.

$\mathsf{TKeygen}$

$P_i(\mathsf{pp})$ All parties $P_{j\neq i}(\mathsf{pp})$

.... Phase 1: CLE Key Generation ....

$\mathsf{CLE.KGen}(\mathsf{pp}) \rightarrow (\mathsf{ek}_i, \mathsf{dk}_i)$
Run $\Sigma_{\mathcal{R}_{\mathsf{key}}}\langle \mathsf{ek}_i, \mathsf{dk}_i\rangle$; Broadcast ek_i

.... Phase 2a: Share Generation ....

$\mathsf{Gen}(\mathsf{pp}) \rightarrow (\{\chi_{ij}, \chi'_{ij}\}_{j\in\mathcal{U}}, C_{\chi_i}, \pi_{\chi_i})$ $\xrightarrow[\text{Send } (\chi_{ij}, \chi'_{ij}) \text{ to } P_{j\neq i}]{\text{Broadcast } (\pi_{\chi_i}, C_{\chi_i})}$ $\mathsf{GenVf}(\chi_{ij}, \chi'_{ij}, C_{\chi_i}, \pi_{\chi_i})$

.... Phase 2b: Share Combination ....

$\mathsf{Comb}(\{\chi_{ji}, \chi'_{ji}\}_{j\in\mathcal{Q}_x}) \rightarrow (w_{x_i}, \mathsf{pub}_{x_i}, \pi_{x_i})$ $\mathsf{CombVf}(\{C_{\chi_j}\}_{j\in\mathcal{Q}_x}, \mathsf{pub}_{x_i}, \pi_{x_i})$

$\mathsf{RevealExp}(w_i, \mathsf{pub}_{x_i}) \rightarrow (X_i, \pi_{X_i})$ $\xrightarrow{\text{Broadcast } (\mathsf{pub}_{x_i}, \pi_{x_i}, X_i, \pi_{X_i})}$ $\mathsf{ExpVf}(\mathsf{pub}_{x_i}, X_i, \pi_{X_i})$

.... Output Phase (See Fig. 4) ....

Fig. 3: Threshold Key Generation Protocol

$\mathsf{TKeygen}$ **(Output Phase)**

parse w_{x_i} **as** (x_i, x'_i, ρ_{x_i})
$\mathsf{vk} := \prod_{j\in\mathcal{U}} X_j^{L_{j,\mathcal{U}}}$; $\mathsf{pk} := \{\mathsf{ek}_j, X_j\}_{j\in\mathcal{U}}$; $\mathsf{sk}_i := (\mathsf{dk}_i, x_i)$
return $(\mathsf{vk}, \mathsf{pk}, \mathsf{sk}_i)$ and erase memory

Fig. 4: Output Phase of Threshold Key Generation Protocol

key generation protocol (Figure 3), pre-signing protocol (Figure 5), and threshold signing protocol (Figures 7-9) by $\mathcal{U}$, $\mathcal{P}$, and $\mathcal{P}^*$. We use the notation $\mathcal{Q}$ in Section IV-B to denote the qualified parties who behaved honestly during the DRG generation phase, *e.g.*, at the end of Phase 2a of $\mathsf{TKeygen}$ ($\mathcal{Q}_x$) and Phase 1a of $\mathsf{PreSign}$ ($\mathcal{Q}_{k,\gamma}$, which fixed k and γ).

Dotted lines in the figures separate a protocol into different phases. Within a phase, the codes on the left-hand side are first executed by party P_i (or party i). The messages above (and below, if any) the arrow will then be sent. Upon receiving P_i's messages, all the other parties execute the (verification) steps on the right-hand side (to verify P_i's messages). Any cheater (irresponsive party included), if identified, is excluded from the participant set. All cheater-related components are discarded. If the participant set remains at least t, the protocol proceeds.

We defer details of cheater identification steps to Section V-D. Table II lists all the variables to be verified.

A. Threshold ECDSA Key Generation

Figure 3 presents our key generation protocol $\mathsf{TKeygen}$. Each party i first generates a CL encryption key-pair $(\mathsf{ek}_i, \mathsf{dk}_i)$ and shows its correctness via an interactive Σ-protocol[13]. If the proof fails to verify, parties exclude the cheater from $\mathcal{U}$.

In Phase 2a, each party i runs $\mathsf{DRG.Gen}$ to distribute share (χ_{ij}, χ'_{ij}) of an initial secret χ_i to party j, and broadcasts its output $c_{\chi_i} := (c_{\chi_i}, F_{\chi_i})$ with a proof π_{χ_i}. Each party j runs $\mathsf{DRG.GenVf}$ to verify the shares from each party i. If it fails, party j broadcasts (χ_{ij}, χ'_{ij}) as a complaint against party i. The cheater is excluded from $\mathcal{U}$ after the complaint is accepted. We mark the qualified set $\mathcal{Q}_x := \mathcal{U}$ for this protocol.

In Phase 2b, parties run $\mathsf{DRG.Comb}$ to obtain (t, n) secret share $\{x_i\}_{i\in\mathcal{Q}_x}$ of the final secret $x := \sum_{i\in\mathcal{Q}_x} \chi_i$. It also outputs public share pub_{x_i}. For later reconstruction of g^x, each party i runs $\mathsf{DRG.RevealExp}$ to reveal $X_i := g^{x_i}$, and finally broadcasts pub_{x_i}, X_i, and ZKPs (π_{x_i}, π_{X_i}) over them. Each party j then runs $\mathsf{DRG.CombVf}$ and $\mathsf{DRG.ExpVf}$ to verify pub_{x_i} and X_i, respectively, and excludes any cheater from $\mathcal{U}$. At this point, correct shares of the initial secret $\chi_{i'}$ of any cheater i' have been sent, so that the protocol can continue.

As in Figure 4, the output to each party i is a threshold-shared ECDSA signing key $\mathsf{sk}_i := (\mathsf{dk}_i, x_i)$ such that $\sum_{i\in\mathcal{U}'} L_{i,\mathcal{U}'} x_i = x$ for any $\mathcal{U}' \subseteq \mathcal{U}$ with $|\mathcal{U}'| \geq t$. It also outputs pk containing CL encryption keys $\{\mathsf{ek}_j = \mathfrak{g}_q^{\mathsf{dk}_j}\}_{j\in\mathcal{U}}$, and $\{X_j = g^{x_j}\}_{j\in\mathcal{U}}$ that defines $\mathsf{vk} := \prod_{j\in\mathcal{U}} X_j^{L_{j,\mathcal{U}}} = g^x$.

Using the key refreshment of DRG, we can achieve proactive security [4] by refreshing x_i underlying the ECDSA key. The only other part is the CL encryption key, which is internal to the signers but not a part of the ECDSA verification key. Each signer can just pick a new key pair as in $\mathsf{TKeygen}$.

B. Offline Pre-signing Protocol

Figure 5 presents our pre-signing protocol $\mathsf{PreSign}$ with three phases: 1) threshold-shared randomness generation using our DRG, 2) share conversion using MtAwc, and 3) share revelation. It aims to generate the following items.

- Pre-signature φ_i for party i including:
  - ○ Share k_i of k; and Protocol nonce $R = g^{1/k}$;
  - ○ Additive shares $\{\mu_{ij}, \nu_{ji}\}_{j\in\mathcal{P}\setminus\{i\}}$ of $k_i x_j = \mu_{ij} + \nu_{ij}$.
- Verification material V_i including:
  - ○ Opening k'_i to commitment PC_{k_i};

[13]$\Sigma_{\mathcal{R}_{\mathsf{key}}}$ is for showing the correctness of the encryption key by generating the challenge collaboratively [9]. We keep it interactive due to our need for the knowledge extractor, which might not be available after the Fiat–Shamir transformation in the multiparty setting [26]. One might use the more (computation and communication) costly NIZK with online extractors [12].

```
PreSign
$P_i(\mathsf{pp}, \mathsf{pk} = \{\mathsf{ek}_j, X_j\}_{j\in\mathcal{P}}, \mathsf{sk}_i = (\mathsf{dk}_i, x_i))$        All parties $P_{j\neq i}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_j)$
........ Phase 1a: Share Generation using DRG ........
$\mathsf{Gen}(\mathsf{pp}) \to (\{\kappa_{ij}, \kappa'_{ij}\}_{j\in\mathcal{P}}, C_{\kappa_i}, \pi_{\kappa_i})$        $\mathsf{GenVf}(\kappa_{ij}, \kappa'_{ij}, C_{\kappa_i}, \pi_{\kappa_i})$
$\mathsf{Gen}(\mathsf{pp}) \to (\{y_{ij}, y'_{ij}\}_{j\in\mathcal{P}}, C_{y_i}, \pi_{y_i})$   $\xrightarrow[\text{Send } (\kappa_{ij}, \kappa'_{ij}, y_{ij}, y'_{ij}) \text{ to } P_{j\neq i}]{\text{Broadcast } (\pi_{\kappa_i}, C_{\kappa_i}, \pi_{y_i}, C_{y_i})}$   $\mathsf{GenVf}(y_{ij}, y'_{ij}, C_{y_i}, \pi_{y_i})$
........ Phase 1b: Share Combination using DRG ........
$\mathsf{Comb}(\{\kappa_{ji}, \kappa'_{ji}\}_{j\in\mathcal{Q}_{k,\gamma}}) \to (w_{k_i}, \mathsf{pub}_{k_i}, \pi_{k_i})$        $\mathsf{CombVf}(\{C_{\kappa_j}\}_{j\in\mathcal{Q}_{k,\gamma}}, \mathsf{pub}_{k_i}, \pi_{k_i})$
$\mathsf{Comb}(\{y_{ji}, y'_{ji}\}_{j\in\mathcal{Q}_{k,\gamma}}) \to (w_{\gamma_i}, \mathsf{pub}_{\gamma_i}, \pi_{\gamma_i})$   $\xrightarrow{\text{Broadcast } (\mathsf{pub}_{k_i}, \pi_{k_i}, \mathsf{pub}_{\gamma_i}, \pi_{\gamma_i})}$   $\mathsf{CombVf}(\{C_{y_j}\}_{j\in\mathcal{Q}_{k,\gamma}}, \mathsf{pub}_{\gamma_i}, \pi_{\gamma_i})$
........ Phase 2: Share Conversion using MtAwc ........
parse $w_{\gamma_i}$ as $(\gamma_i, \gamma'_i, \rho_{\gamma_i})$
$\mathsf{RevealExp}(w_{\gamma_i}, \mathsf{pub}_{\gamma_i}) \to (\Gamma_i, \pi_{\Gamma_i})$   $\xrightarrow[\text{Broadcast } \{c_{\alpha_{ji}}, \mathcal{B}_{ji}, c_{\mu_{ji}}, \mathcal{N}_{ji}\}_j]{\text{Broadcast } (\Gamma_i, \pi_{\Gamma_i})}$   $\mathsf{ExpVf}(\mathsf{pub}_{\gamma_i}, \Gamma_i, \pi_{\Gamma_i})$
foreach $j \in \mathcal{P} \setminus \{i\}$ do
  parse $\mathsf{pub}_{k_j}$ as $(\mathsf{PC}_{k_j}, c_{k_j})$
  $\beta_{ji}, \nu_{ji} \leftarrow\$ \mathbb{F}_q$; $\mathcal{B}_{ji} := g^{\beta_{ji}}$; $\mathcal{N}_{ji} := g^{\nu_{ji}}$
  $c_{\alpha_{ji}} \leftarrow \gamma_i \otimes c_{k_j} \oplus \mathsf{Enc}(\mathsf{ek}_j, -\beta_{ji})$        $\alpha_{ji} := \mathsf{Dec}(\mathsf{dk}_j, c_{\alpha_{ji}})$; check $g^{\alpha_{ji}}\mathcal{B}_{ji} \overset{?}{=} \Gamma_i^{k_j}$
  $c_{\mu_{ji}} \leftarrow x_i \otimes c_{k_j} \oplus \mathsf{Enc}(\mathsf{ek}_j, -\nu_{ji})$        $\mu_{ji} := \mathsf{Dec}(\mathsf{dk}_j, c_{\mu_{ji}})$; check $g^{\mu_{ji}}\mathcal{N}_{ji} \overset{?}{=} X_i^{k_j}$
endforeach
........ Phase 3: Share Revelation ........
parse $w_{k_i}$ as $(k_i, k'_i, \rho_{k_i})$
$\Gamma := \prod_{j\in\mathcal{P}} \Gamma_j^{L_{j,\mathcal{P}}}$; $D_i := \Gamma^{k_i}$
$\pi_{D_i} \leftarrow \mathsf{NIZK}_{\mathsf{DL\text{-}PC}}.\mathsf{Prove}((\mathsf{PC}_{k_i}, \Gamma, D_i); (k_i, k'_i))$   $\xrightarrow{\text{Broadcast } (\pi_{D_i}, D_i, \{\delta_{ij}\}_j)}$   $\mathsf{NIZK}_{\mathsf{DL\text{-}PC}}.\mathsf{Verify}((\mathsf{PC}_{k_i}, \Gamma, D_i), \pi_{D_j})$
$\{\theta_{ij}\}_{j\in\mathcal{P}} \leftarrow \mathsf{SS.Share}(0)$; $\delta_{ii} := k_i\gamma_i + \theta_{ii}$        $\delta_{i,\mathcal{P}} := \sum_{\ell\in\mathcal{P}} L_{\ell,\mathcal{P}}\delta_{j\ell}$
for $(j \in \mathcal{P} \setminus \{i\})$ do $\{\delta_{ij} := \alpha_{ij} + \beta_{ji} + \theta_{ij}\}$        check $g^{\delta_{i,\mathcal{P}}} \prod_{\ell\in\mathcal{P}\setminus\{i\}} (\mathcal{B}_{i\ell}/\mathcal{B}_{\ell i})^{L_{\ell,\mathcal{P}}} \overset{?}{=} D_i$
........ Output Phase: Nonce Reconstruction (See Fig. 6) ........
```

Fig. 5: Pre-signing Protocol

```
PreSign (Output Phase)
  $\{\delta_{j,\mathcal{P}}\}_{j\in\mathcal{P}} := \{\sum_{\ell\in\mathcal{P}} L_{\ell,\mathcal{P}}\delta_{j\ell}\}_{j\in\mathcal{P}}$
  $\delta := \sum_{j\in\mathcal{P}} L_{j,\mathcal{P}}\delta_{j,\mathcal{P}}$
  $R := \Gamma^{1/\delta}$;   $\{R_j\}_{j\in\mathcal{P}} := \{D_j^{1/\delta}\}_{j\in\mathcal{P}}$
  $\varphi_i := (R, k_i, \{\mu_{ij}, \nu_{ji}\}_{j\in\mathcal{P}\setminus\{i\}})$
  $V_i := (k'_i, \{\mathsf{PC}_{k_j}\}_{j\in\mathcal{P}}\{R_j\}_{j\in\mathcal{P}}, \{\{\mathcal{N}_{j\ell}\}_{\ell\in\mathcal{P}\setminus\{j\}}\}_{j\in\mathcal{P}})$
  return $(\varphi_i, V_i)$ and erase memory
```

Fig. 6: Output Phase of Pre-signing for Nonce Reconstruction

- Commitments $\{\mathsf{PC}_{k_j}\}_{j\in\mathcal{P}}$ of shares $\{k_j\}_{j\in\mathcal{P}}$;
- Shares $\{R_j = R^{k_j}\}_{j\in\mathcal{P}}$ of k; and Additive shares $\{\{\mathcal{N}_{j\ell}\}_{\ell\in\mathcal{P}\setminus\{j\}} = \{g^{\nu_{j\ell}}\}_{\ell\in\mathcal{P}\setminus\{j\}}\}_{j\in\mathcal{P}}$, both in exponent.

1) Phase 1: Party i runs DRG.Gen(pp) to generate its initial secrets κ_i, y_i and sends their verifiable shares to each party via private channels. Party i broadcasts C_{κ_i}, C_{y_i} containing ciphertexts c_{κ_i}, c_{y_i} of κ_i, y_i and their proofs $\pi_{\kappa_i}, \pi_{y_i}$.

Party j verifies shares (κ_{ij}, y_{ij}) from party i against the broadcasted verification materials (C_{κ_i}, C_{y_i}) by party i, If it fails, party j broadcasts a complaint to exclude party i from $\mathcal{P}$. At the end of Phase 1a, each party i holds the correct shares $\{\kappa_{ji}, y_{ji}\}_{j\in\mathcal{P}}$ of initial secrets $\{\kappa_j, y_j\}_{j\in\mathcal{P}}$. We define the qualified set $\mathcal{Q}_{k,\gamma} := \mathcal{P}$ at this point.

Phase 1b runs the combination phase of DRG to generate threshold secret shares $\{k_i, \gamma_i\}_{i\in\mathcal{P}}$ of secrets $k := \sum_{i\in\mathcal{Q}_{k,\gamma}} k_i$ and $\gamma := \sum_{i\in\mathcal{Q}_{k,\gamma}} \gamma_i$. Party i runs DRG.Comb to combine the verified and received shares $\{\kappa_{ji}, y_{ji}\}_{i\in\mathcal{Q}_{k,\gamma}}$, into k_i, γ_i, and their public shares $\mathsf{pub}_{k_i}, \mathsf{pub}_{\gamma_i}$, which contains commitments $\mathsf{PC}_{k_i}, \mathsf{PC}_{\gamma_i}$. and ciphertexts c_{k_i}, c_{γ_i}. Party i broadcasts them with proofs $(\pi_{k_i}, \pi_{\gamma_i})$ for $\mathcal{R}_{\mathsf{Enc\text{-}PC}}$ that the commitments and ciphertexts are either committing to or encrypting the same (k_i, γ_i). Party j verifies the proof $(\pi_{k_i}, \pi_{\gamma_i})$ about $(\mathsf{pub}_{k_i}, \mathsf{pub}_{\gamma_i})$. If it fails, party j excludes party i from $\mathcal{P}$.

At the end, secret output w_{γ_i} for party i consists of secret share γ_i, encryption randomness ρ_{γ_i}, and similar items for w_{k_i}. Each party i also holds commitments $\{\mathsf{PC}_{k_j}, \mathsf{PC}_{\gamma_j}\}_{j\in\mathcal{P}}$ and ciphertexts $\{c_{k_j}, c_{\gamma_j}\}_{j\in\mathcal{P}}$ of $\{k_j, \gamma_j\}_{j\in\mathcal{P}}$.

2) Phase 2: For shares $\{k_i, \gamma_i, x_i\}_{i\in\mathcal{P}}$ of secrets (k, γ, x), we aim to convert $k_j\gamma_i$ and k_jx_i such that $k_j\gamma_i = \alpha_{ji} + \beta_{ji}$ and $k_jx_i = \mu_{ji} + \nu_{ji}$, where party i holds $(\gamma_i, x_i, \beta_{ji}, \nu_{ji})$ and party j holds $(k_j, \alpha_{ji}, \mu_{ji})$. Furthermore, we generate the shares in exponent $\{\{\mathcal{B}_{j\ell}, \mathcal{N}_{j\ell}\}_{j\in\mathcal{P}\setminus\{\ell\}}\}_{\ell\in\mathcal{P}} = \{\{g^{\beta_{j\ell}}, g^{\nu_{j\ell}}\}_{j\in\mathcal{P}\setminus\{\ell\}}\}_{\ell\in\mathcal{P}}$ and $\{\Gamma_j\}_{j\in\mathcal{P}} = \{g^{\gamma_j}\}_{j\in\mathcal{P}}$.

When Phase 2 starts, party i computes and broadcasts $\Gamma_i = g^{\gamma_i}$ with proof that it is bound to the commitment (in pub_{γ_i}) broadcasted during Phase 1b. Phase 2 runs MtAwc, where party i (acting as a sender) and j computes the additive share of $k_jx_i, k_j\gamma_i$ with the broadcasted ciphertext c_{k_j} from party j in Phase 1b, *i.e.*, parties j and i engage in $\mathsf{MtAwc}\langle(\Gamma_i, k_j, (\mathsf{ek}_j, \mathsf{dk}_j)); (\Gamma_i, \gamma_i, \mathsf{ek}_j)\rangle \to \langle(g^{\beta_{ji}}, \alpha_{ji}); (\beta_{ji})\rangle$ to convert $k_j\gamma_i$ into $\alpha_{ji} + \beta_{ji}$.

Party $i \in \mathcal{P}$ owning private shares x_i, γ_i can homomorphically compute the ciphertext of $\alpha_{ji} = \gamma_i \cdot k_j - \beta_{ji}$ and $\mu_{ji} = x_i \cdot k_j - \nu_{ji}$ for randomly sampled β_{ji}, ν_{ji}. The ciphertexts $c_{\alpha_{ji}}, c_{\mu_{ji}}$ and the shares in exponent $\mathcal{B}_{ji} = g^{\beta_{ji}}, \mathcal{N}_{ji} = g^{\nu_{ji}}$ are broadcasted alongside the proof on Γ_i.

Party j decrypts and checks the received shares α_{ji}, μ_{ji} with $\Gamma_i, \mathcal{B}_{ji}, k_i$, *e.g.*, $g^{\alpha_{ji}}\mathcal{B}_{ji} \stackrel{?}{=} \Gamma_i^{k_j}$. To complain about an incorrect share, *e.g.*, α_{ji}, party j broadcasts proof of decryption of $c_{\alpha_{ji}}$ and proof that the exponent in $\Gamma_i^{k_j}$ is bound to PC_{k_j}. Party i will be removed from $\mathcal{P}$ if the complaint is valid. The complaint for μ_{ji} works similarly.

At the end of Phase 2, each party j stores additive shares $\{\alpha_{ji}, \beta_{ij}, \mu_{ji}, \nu_{ij}\}_{i\in\mathcal{P}}$. Every party also stores the public values $\{\mathcal{B}_{i\ell}, \mathcal{N}_{i\ell}\}_{(i,\ell)\in\mathcal{P}^2, i\neq\ell}$ and $\{\Gamma_i = g^{\gamma_i}\}_{i\in\mathcal{P}}$.

3) Phase 3: We aim to broadcast shares δ_{ij} of $\delta = k\gamma$ securely. Since $\{\Gamma_j = g^{\gamma_j}\}_{j\in\mathcal{P}}$ are published and verified in Phase 2, the parties compute Γ by Lagrange interpolation in exponent, *i.e.*, $\Gamma = \prod_{j\in\mathcal{P}} \Gamma_j^{L_{j,\mathcal{P}}}$. Thus, party i holding k_i can also compute $D_i = \Gamma^{k_i} = g^{k_i\gamma}$. This D_i will be broadcasted with proof of committed exponent ($\mathcal{R}_{\mathsf{DL\text{-}PC}}$) of Γ with k_i committed in PC_{k_i} from Phase 1b. Note that $D_i = g^{k_i \cdot \sum_{j\in\mathcal{P}}(L_{j,\mathcal{P}}\gamma_j)}$ can be used to verify the correctness of $k_i \cdot \sum_{j\in\mathcal{P}}(L_{j,\mathcal{P}}\gamma_j)$ in exponent. We aim to use D_i and $\{\mathcal{B}_{ji}\}_{i,j}$ to verify the shares $\{\delta_{ij}\}_{j\in\mathcal{P}}$ of pseudo-nonce.

In Phase 3, party i releases their threshold shares $\{\delta_{ij}\}_{j\in\mathcal{P}}$ of the pseudo-nonce $k\gamma$, where

$$\delta_{ij} = \begin{cases} \alpha_{ij} + \beta_{ji} + \theta_{ij}, & \text{if } i \neq j \\ k_i\gamma_i + \theta_{ii}, & \text{if } i = j \end{cases}$$

and $\{\theta_{ij}\}_{j\in\mathcal{P}}$ is Shamir sharing of 0. To illustrate, assume $\mathcal{P} = \{1, \ldots, n\}$ and consider the following $n \times n$ matrix $\mathbf{S}$:

$$\mathbf{S} = \begin{pmatrix} k_1\gamma_1 & \alpha_{12} + \beta_{21} & \cdots & \alpha_{1n} + \beta_{n1} \\ \alpha_{21} + \beta_{12} & k_2\gamma_2 & \cdots & \alpha_{2n} + \beta_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{n1} + \beta_{1n} & \cdots & \cdots & k_n\gamma_n \end{pmatrix}$$

where row i is the "unmasked version" of $\{\delta_{ij}\}_{j\in\mathcal{P}}$. Each δ_{ij} is the sum of party i's share of $(k_i\gamma_j, k_j\gamma_i)$ obtained in the previous phase after interacting with parties $j \in \mathcal{P} \setminus \{i\}$. Multiplying masked row i with $\vec{\ell} = (L_{1,n}, L_{2,n}, \ldots, L_{n,n})^\intercal$ gives $\delta_{i,\mathcal{P}} = (L_{1,n}(\alpha_{i1}+\beta_{1i})+\cdots+L_{i,n}k_i\gamma_i+\cdots+L_{n,n}(\alpha_{in}+\beta_{ni}))$ where the 0 mask is canceled.

To verify the broadcasted share, consider matrix $\mathbf{T}$ below:

$$\mathbf{T} = \begin{pmatrix} 0 & \beta_{12} - \beta_{21} & \cdots & \beta_{1n} - \beta_{n1} \\ \beta_{21} - \beta_{12} & 0 & \cdots & \beta_{2n} - \beta_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{n1} - \beta_{1n} & \cdots & \cdots & 0 \end{pmatrix}.$$

We have $\mathbf{S} \cdot \vec{\ell} + \mathbf{T} \cdot \vec{\ell} = (k_1\gamma, k_2\gamma, \ldots, k_n\gamma)^\intercal$.

The above check can be performed over the exponent by linear homomorphism since the Lagrange coefficients and masked shares $\mathbf{S}$ are known. Lifting $\mathbf{T}$ to group generator g:

$$g^{\mathbf{T}} = \begin{pmatrix} g^0 & g^{\beta_{12}-\beta_{21}} & \cdots & g^{\beta_{1n}-\beta_{n1}} \\ g^{\beta_{21}-\beta_{12}} & g^0 & \cdots & g^{\beta_{2n}-\beta_{n2}} \\ \vdots & \vdots & \ddots & \vdots \\ g^{\beta_{n1}-\beta_{1n}} & \cdots & \cdots & g^0 \end{pmatrix},$$

we have $g^{\beta_{ij}-\beta_{ji}} = \mathcal{B}_{ij}/\mathcal{B}_{ji}$, which can be computed using $\{\mathcal{B}_{ji}\}_{i,j}$ from Phase 2. Thus, the parties can compute $g^{k_i\gamma}$ in exponent, using the outputs from MtAwc and broadcasted shares $\{\delta_{ij}\}_{j\in\mathcal{P}}$ by party i to check against $D_i = \Gamma^{k_i}$:

$$g^{\delta_{i,\mathcal{P}}} \prod_{j\in\mathcal{P}\setminus\{i\}} (\mathcal{B}_{ij}/\mathcal{B}_{ji})^{L_{j,\mathcal{P}}} \stackrel{?}{=} D_i.$$

4) Nonce Reconstruction: Given $\{\{\delta_{j\ell}\}_{\ell\in\mathcal{P}}\}_{j\in\mathcal{P}}$, party i first reconstructs $\delta = k\gamma$ by performing two (iterations of) Lagrange interpolations on the shares[14] as in Figure 6. Finally, our pre-signing protocol outputs two types of values:

1) The pre-signature (share) φ_i for online signing, including the nonce of signature $R = g^{1/k}$, computed by $\Gamma^{1/\delta} = g^{\gamma/\delta}$, the partial randomness k_i, and additive shares $\{(\mu_{ij}, \nu_{ji})\}_{j\in\mathcal{P}\setminus\{i\}}$;
2) The verification materials V_i for cheater identification in online signing, which contain the party's commitment randomness k_i', commitments $\{\mathsf{PC}_{k_j}\}_{j\in\mathcal{P}}$ to shares k_j from others, share k_j in exponent (to base R) $\{R_j\}_{j\in\mathcal{P}}$ computed by $D_j^{1/\delta}$ and all MtA shares of the pseudo-key $\mathcal{N}_{ij} = g^{\nu_{ij}}$.

C. Online Signing Protocol

TSign aims to generate (r, s) such that $s = km + rkx$ or equivalently $R^s = g^mX^r$, where $m = H(\mathsf{msg})$, $X = g^x$ is the verification key, and r is the x-projection of $R = g^{1/k}$. Each party $i \in \mathcal{P}^*$ builds upon party i's pre-signature (φ_i, V_i):

1) Party i runs Phase 1a in Figure 7 to broadcast threshold signature shares $\{s_{ij}\}_{j\in\mathcal{P}^*}$ and verification material $\mathbb{V}_i$:
   - Computing s_{ij} is similar to δ_{ij} in Phase 3 of $\mathsf{PreSign}$, with k_i times the j-th share of m as the "mask." Shares of m created by party i are different from those by others.
   - Verification materials $\mathbb{V}_i$ are essentially the additive shares in exponent to the base element R, with ZKP.
2) Upon receiving $\{s_{ij}\}_{j\in\mathcal{P}^*}$ from party i, party j runs Phase 1b in Figure 8 to verify its correctness. Party j first verifies the additive share in exponent to base R via ZKP, then verifies the threshold signature $\{s_{ij}\}_{j\in\mathcal{P}^*}$ using the

[14]Recall that $k\gamma = \sum_{i,j\in[n]} L_iL_jk_i\gamma_j = \sum_{i\in[n],j<i} L_iL_j(k_i\gamma_j + k_j\gamma_i) + \sum_{i\in[n]} L_i^2k_i\gamma_i$, with $\delta_{ij} + \delta_{ji} = k_i\gamma_j + k_j\gamma_i$, the interpolation is essentially multiplying the matching Lagrange coefficients. With the matrix notation, we have $\mathbf{S}[i,j] + \mathbf{S}[j,i] = k_i\gamma_j + k_j\gamma_i$ and $k\gamma = (\vec{\ell})^\intercal \cdot \mathbf{S} \cdot \vec{\ell}$.

```
TSign (Phase 1a)
$P_i(\mathsf{pp}, \{\mathsf{ek}_j, X_j\}_{j\in\mathcal{P}^*}, (\mathsf{dk}_i, x_i), (\varphi_i, V_i), \mathsf{msg})$
parse $\varphi_i$ as $(R, k_i, \{\mu_{ij}, \nu_{ji}\}_{j\in\mathcal{P}^*\setminus\{i\}})$
parse $V_i$ as $\big(k'_i, \{\mathsf{PC}_{k_j}\}_{j\in\mathcal{P}^*}, \{R_j\}_{j\in\mathcal{P}^*},$
          $\{\{\mathcal{N}_{j\ell}\}_{\ell\in\mathcal{P}^*\setminus\{j\}}\}_{j\in\mathcal{P}^*}\big)$
foreach $j \in \mathcal{P}^* \setminus \{i\}$ do
  $\mathcal{M}_{ij} := R^{\mu_{ij}}$
  $\pi_{\mathcal{M}_{ij}} \leftarrow \mathsf{NIZK}_{\mathsf{DL\text{-}2PC}}.\mathsf{Prove}($
      $(\mathsf{PC}_{k_i}, X_j, 1/g, \mathcal{N}_{ij}, R, \mathcal{M}_{ij}); (k_i, k'_i, \mu_{ij}))$
endforeach
$m := \mathsf{H}(\mathsf{msg});\ \{m_{ij}\}_{j\in\mathcal{P}^*} \leftarrow \mathsf{SS.Share}(m)$
Let $r$ be the x-projection of EC-point $R$; $s_{ii} := rk_ix_i + k_im_{ii}$
foreach $(j \in \mathcal{P}^* \setminus \{i\})$ do $\{s_{ij} := r(\mu_{ij} + \nu_{ji}) + k_im_{ij}\}$
Broadcast $\{s_{ij}\}_{j\in\mathcal{P}^*}, \mathbb{V}_i := \{\mathcal{M}_{ij}, \pi_{\mathcal{M}_{ij}}\}_{j\in\mathcal{P}^*\setminus\{i\}}$
```

Fig. 7: Threshold Signature Shares Revelation

```
TSign (Phase 1b)
All parties $P_{j\neq i}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_j, (\varphi_j, V_j), \mathsf{msg})$
foreach $\ell \in \mathcal{P}^* \setminus \{i\}$ do
  $\mathsf{NIZK}_{\mathsf{DL\text{-}2PC}}.\mathsf{Vf}((\mathsf{PC}_{k_i}, X_\ell, 1/g, \mathcal{N}_{i\ell}, R, \mathcal{M}_{i\ell}), \pi_{\mathcal{M}_{i\ell}})$
$s_{i,\mathcal{P}^*} := \sum_{\ell\in\mathcal{P}^*} L_{\ell,\mathcal{P}^*} s_{i\ell}$
check $R^{s_{i,\mathcal{P}^*}} \prod_{\ell\in\mathcal{P}^*\setminus\{i\}} (\mathcal{M}_{\ell i}/\mathcal{M}_{i\ell})^{rL_{i,\mathcal{P}^*}} \overset{?}{=} R_i^m X_i^r$
```

Fig. 8: Threshold Signature Shares Combination

```
TSign (Phase 2)
$P_i(\{s_{j\ell}\}_{j,\ell\in\mathcal{P}^*})$
$\{s_{j,\mathcal{P}^*}\}_{j\in\mathcal{P}^*} := \{\sum_{\ell\in\mathcal{P}^*} L_{\ell,\mathcal{P}^*} s_{j\ell}\}_{j\in\mathcal{P}^*}$
return $(r, s := \sum_{j\in\mathcal{P}^*} L_{j,\mathcal{P}^*} s_{j,\mathcal{P}^*})$
```

Fig. 9: Threshold Signature Reconstruction

additive share in exponent. The verification logic is similar to the one in Phase 3 of $\mathsf{PreSign}$ (*cf.* Section V-B3) for $\{\delta_{ij}\}_j$, *e.g.*, $\mathbf{S}[i,j] = \mu_{ij} + \nu_{ji}$ (k_ix_i) for $i \neq j$ $(i = j)$ with $k_i \cdot \mathsf{SS.Share}(m)$ as the (row i) "mask," and D_i is replaced by $R_i^m X_i^r$. If any verification fails against party i's broadcasted shares, party j excludes the cheater from $\mathcal{P}^*$.

3) After completing the verification, party i runs Phase 2 in Figure 9 to reconstruct $s := km+rkx$ of the ECDSA signature $\sigma := (r, s)$ by Lagrange interpolation. We emphasize that $\{s_{i,\mathcal{P}^*}\}_{i\in\mathcal{P}^*}$ computed during Phases 1b and 2 may differ because the value depends on the updating set $\mathcal{P}^*$ (some parties may kick out after Phase 1b).

Unforgeability will be analyzed in Appendix C-B.

D. Cheater Identification

We describe the cheater identification for each phase, which shows that our scheme achieves identifiable abort under the dishonest majority setting. The robustness under the honest majority setting is deferred to Appendix C-A.

TABLE II: Data Verified by Party i in Chronological Order

Variable	Meaning/Origin	Verified by
ek_j	CLE encryption key of party j	$\mathsf{NIZK}_{\mathsf{key}}$
χ_j, κ_j, y_j	Contribution by party j for x and $k\gamma$	$\mathsf{DRG.GenVf}$
$\chi_{ji}, \chi'_{ji}, (F_{\chi_j}, c_{\chi_j})$ (resp. κ_{ji}, y_{ji})	VSS share of (χ_j, κ_j, y_j) by party j for party i (and verification material)	VSS
$\mathsf{PC}_{x_j}, c_{x_j}$ (resp. k_j, γ_j)	VSS commitment or ciphertext of x_j from party j	$\mathsf{DRG.CombVf}$
$X_j = g^{x_j}, \Gamma_j = g^{\gamma_j}$	Combined share in exponent from party j	$\mathsf{DRG.ExpVf}$
α_{ji}, μ_{ji}	P_i's decrypted MtAwc shares for $k_j\gamma_i, k_jx_i$	MtAwc
$\mathcal{B}_{ji}, \mathcal{N}ji$	P_j's MtAwc share-in-exponent for $k_j\gamma_i, k_jx_i$	MtAwc
$D_j = \Gamma^{k_j}$	Broadcasted share k_i in exponent	$\mathsf{NIZK}_{\mathsf{DL\text{-}PC}}$
$\{\delta_{j\ell}\}_{\ell\in\mathcal{P}}$	Unstructured shares of $k\gamma$	Equation (2)
$\mathcal{M}_{ji}$	P_i's MtAwc shares for k_jx_i in exponent	$\mathsf{NIZK}_{\mathsf{DL\text{-}2PC}}$
$\{s_{j\ell}\}_{\ell\in\mathcal{P}}$	Unstructured shares of $s = km + rkx$	Equation (3)

1) Identification in $\mathsf{TKeygen}$*:* Let $\mathcal{U}$ be the set of participants in $\mathsf{TKeygen}$. In Phase 1, each party verifies ZKP for $\mathcal{R}_{\mathsf{key}}$, asserting that the CLE key pair satisfies $\mathsf{ek}_i = \mathfrak{g}_q^{\mathsf{dk}_i}$. If it is invalid, party i is deemed a cheater by updating $\mathcal{U}$ to $\mathcal{U}\setminus\{i\}$.

In Phase 2, the remaining honest parties in $\mathcal{U}$ engage in the DRG protocol to generate the shares of the threshold ECDSA key $(\{x_j\}_{j\in\mathcal{U}})$ and the public shares $\{\mathsf{pub}_{x_j}\}_{j\in\mathcal{U}}$ $(\{X_j = g^{x_j}\}_{j\in\mathcal{U}})$. The cheater identification for invalid keys and public shares largely follows DRG (*cf.*, Section IV-B).

2) Identification in $\mathsf{PreSign}$*:* Let $\mathcal{P}$ be the set of participants in $\mathsf{PreSign}$. Phase 1 runs DRG to share k, γ. Similar to $\mathsf{TKeygen}$, the parties can identify and remove cheating parties.

For MtAwc (Phase 2 in Figure 5), only party j can decrypt the ciphertexts $(c_{\alpha_{ij}}, c_{\mu_{ij}})$ and verify the decrypted shares (α_{ij}, μ_{ij}) with $(\mathcal{B}_{ij}, \mathcal{N}_{ij})$ broadcasted by party i. To complain about a cheating party i, party j broadcasts α_{ij} with proof $\mathcal{R}_{\mathsf{Dec}}$ of decryption of $c_{\alpha_{ij}}$ and $\Gamma_i^{k_j}$ (resp. $X_i^{k_j}$) with proof $\mathcal{R}_{\mathsf{DL\text{-}PC}}$ of k_j in $\Gamma_i^{k_j}$ committed in PC_{k_j} to the base Γ_i. If the proofs and components $(\Gamma_i^{k_j}, \mathcal{B}_{ij}, \alpha_{ij})$ (resp. $(X_i^{k_j}, \mathcal{N}_{ij}, \mu_{ij})$) broadcasted by party j are valid, others can verify α_{ij} (resp. μ_{ij}) by checking $g^{\alpha_{ij}}\mathcal{B}_{ij} \overset{?}{=} \Gamma_i^{k_j}$ (resp. $g^{\mu_{ij}}\mathcal{N}_{ij} \overset{?}{=} X_i^{k_j}$).

In Phase 3, parties verify the share in exponent $D_i = \Gamma^{k_i} = g^{\gamma k_i}$ to the base element Γ and the shares $\{\delta_{ij}\}_{j\in\mathcal{P}}$ of pseudo-nonce $\delta = k\gamma$. While ZKP directly ensures the correctness of D_i, verifying $\{\delta_{ij}\}_{j\in\mathcal{P}}$ is more complicated.

The verification equation verifies the shares $\{\delta_{ij}\}_{j\in\mathcal{P}}$ from party i as $\delta_{i,\mathcal{P}} = L_{i,\mathcal{P}}k_i\gamma_i + \sum_{j\in\mathcal{P}\setminus\{i\}} L_{j,\mathcal{P}}(\alpha_{ij} + \beta_{ji})$ in exponent (*cf.*, Section V-B3 for the correctness of Equation (2)):

$$g^{\delta_{i,\mathcal{P}}} \prod_{j\in\mathcal{P}\setminus\{i\}} (\mathcal{B}_{ij}/\mathcal{B}_{ji})^{L_{j,\mathcal{P}}} \overset{?}{=} D_i. \tag{2}$$

All parties exclude party i from $\mathcal{P}$ if the broadcast share $\{\delta_{ij}\}_{j\in\mathcal{P}}$ fail to satisfy Equation (2). Since $\{\mathcal{B}_{ij}, \mathcal{B}_{ji}\}$ have been verified in Phase 2 and the well-formedness of D_i is ensured by ZKP, the party can verify the shares $\{\delta_{ij}\}_{j\in\mathcal{P}}$ from party i. Although the verification involves $\{\mathcal{B}_{ji}\}_{j\in\mathcal{P}\setminus\{i\}}$ that party j generates, party i has verified $\{\mathcal{B}_{ij}\}_{j\in\mathcal{P}\setminus\{i\}}$ and reported the fault. Thus, any fault identified in $\delta_{i,\mathcal{P}}$ can always be imputed to party i. Even if the fault stems from incorrect $\mathcal{B}_{ji}$ broadcasted by party j during MtAwc, party i is still held responsible for the unreported culprit of $\mathcal{B}_{ji}$ in Phase 2.

TABLE III: Costs of Zero-Knowledge Proofs: G, F, QF denotes the communication and computation costs for the EC group $\mathbb{G}$, field $\mathbb{F}_q$ and the binary quadratic form QF, resp.

ZKP Relation	Comm.	Prover Comp.	Verifier Comp.
PC	$3F + 1G$	$2G$	$3G$
DL-PC	$3F + 2G$	$3G$	$5G$
DL-2PC	$4F + 3G$	$5G$	$8G$
Enc-PC	$4F + 1G + 6QF$	$2G + 5QF$ (185 ms)	$3G + 15QF$ (89.7 ms)
Dec	$4F + 6QF$	$6QF$ (206 ms)	$12QF$ (87.5 ms)

TABLE IV: Communication Costs of the Protocols/Primitives of Our (t, n) Threshold ECDSA Scheme: DRG^* runs $(\mathsf{RevealExp}, \mathsf{ExpVf})$, and DRG does not.

Protocol/	Comm. Costs		
Primitives	P2P	Broadcast	Comp. Cost
DRG*	$2F$	$10F + (t+3)G + 16QF$	$(4t + 34n)G + (30n)QF$
DRG	$2F$	$8F + (t+2)G + 16QF$	$(4t + 30n)G + (30n)QF$
TKeygen	$2F$	$13F + (t+5)G + 16QF$	$(4t + 35n)G + (32n)QF$
PreSign	$4F$	$(16+t)F + (6t+9)G$ $+(2n+32)QF$	$(8t + 18n)G + (32n)QF$
TSign	-	$4(n-1)F + (6n-5)G$	$(8n^2 + 6n)G$

TABLE V: Local Computation Time and Total Incoming Communication Cost ($n = 5, t = 4$), excluding extra identification costs; DRG, MtA, Reveal corresponds to Phase 1, 2-3, and 4-6 of the CL-based scheme, respectively.

Steps	RunTime (ms)	Comm. Costs (Bytes)	CL-based [7] Comm.	Paillier-based [4] Comm.
Pre-sign (DRG)	2503	26536	11920	18496
Pre-sign (MtA)	1365	7584	7804	80464
Pre-sign (Reveal)	6	2992	6164	14568
Online	10	9236	240	285

3) Identification in TSign*:* With the correct additive share in exponent $\mathcal{M}_{ij} = R^{\mu_{ij}}$ (ensured by ZKP), we follow the verification steps for δ_{ij} to verify the threshold signatures $\{s_{ij}\}_{j \in \mathcal{P}^*}$ from party i as $s_{i,\mathcal{P}^*} = k_i m + r(L_{i,\mathcal{P}^*} k_i x_i + \sum_{j \in \mathcal{P}^* \setminus \{i\}} L_{j,\mathcal{P}^*}(\mu_{ij} + \nu_{ji}))$:

$$R^{s_{i,\mathcal{P}^*}} \prod_{j \in \mathcal{P}^* \setminus \{i\}} (\mathcal{M}_{ji}/\mathcal{M}_{ij})^{r L_{j,\mathcal{P}^*}} \overset{?}{=} R_i^m X_i^r. \tag{3}$$

For any fault due to $\{\mathcal{N}_{ij}, \mathcal{N}_{ji}\}$ when verifying ZKP of $\{\mathcal{M}_{ij}, \mathcal{M}_{ji}\}$, we can safely impute the fault to party i, by the similar argument for $\{\mathcal{B}_{ij}, \mathcal{B}_{ji}\}$. Notice that (R_i, X_i) are guaranteed to be correct in $\mathsf{TKeygen}$ and $\mathsf{PreSign}$, respectively, so the party can verify the shares $\{s_{ij}\}_{j \in \mathcal{P}^*}$ from party i.

VI. Efficiency Analysis and Implementation

A. Efficiency Analysis

Our protocol employs CL encryption and ZKPs listed in Section III-F (instantiated in Appendix A-B). The elements of the class group are represented in binary quadratic forms, and we denote by QF the group element size and the group operation. The letter G represents the element size and the group operation cost in the EC group; F represents the element size in $\mathbb{F}_q$ (computational cost is neglected). Table III lists the proof size and verification cost, and Table IV lists the communication and computational costs of sub-protocols.

B. Implementation

We benchmark our protocol using the ZenGo-X libraries[15] with class-group and elliptic-curve libraries[16]. We use the Secp256k1 curve and set $\lambda = 128$ ($|\Delta_q| = 1860$). We use a desktop computer equipped with AMD Ryzen 7 3700X (at 4GHz) running Windows Subsystem for Linux (Ubuntu 20.04) allocated with 75GB RAM. Table V lists the timings for subprotocols and the total inbound communication cost per party, including the JSON-serialization overhead.

We also provide the communication cost for two IA schemes: the 6-round CL-based "CCLST23" [7] and the 3-round Paillier-based "CGGMP20" [4] under the same set of parameters and with compact ZKPs for the CL encryption [27]. We do not include a benchmark with SMC-based protocol [18] for not achieving IA in the pre-signing phase. Besides, we also do not compare to Damgård *et al.* [10] and Pettit [24] since they require $\geq 2t + 1$ participants in the pre-signing phase.

CGGMP20 [4] has a faster running time since it only requires EC operations and exponentiation in rings $\mathbb{Z}_N, \mathbb{Z}_{N^2}$, but with $3\times$ communication costs of CCLST23 and ours.

If no one cheats, ours has higher communication costs (36.2KB vs. 25.3KB vs. 110KB) in the pre-signing protocol for $(t, n) = (4, 5)$. If an error occurs during MtA or share revelation, ours allows continuation after identifying the cheaters and thus saves $\sim$21%$-$32% of total communication cost compared to CCLST23 (*cf.* Figures 10a-10b). Similarly, our total runtime is $\sim$30% higher than CCLST23 for $(n-1, n)$, and $\sim$10%$-$30% lower in case of an error in Phase 3 (*cf.* Figure 11).

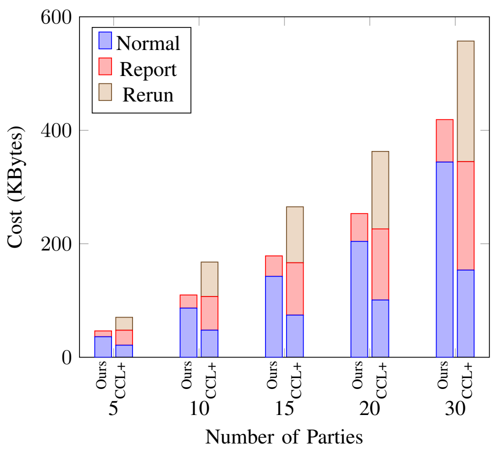
The higher cost comes from verifably encrypting the secret input in our DRG. In our online signing protocol, the parties can identify and discard malformed shares at the cost of $\mathcal{O}(n)$ group elements. To compute the threshold signature after identifying cheaters, the remaining (honest) parties do not need to run the pre-signing protocol again. For schemes that do not self-heal, the performed computation and communication are the costs to deal with cheaters. For CCLST23, any abort occurs at the pre-signing during MtA will waste $\sim$50% of the work.

VII. Conclusion

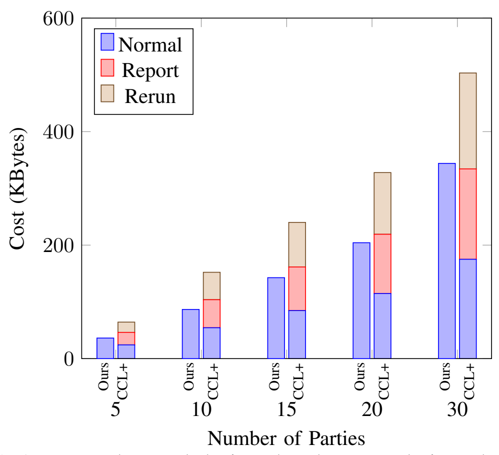
Many threshold ECDSA proposals focus on minimizing communication costs but neglect to ensure that a protocol run will not abort in the presence of faulty signers. In practice, such weakness wastes the communication and computation efforts of the remaining honest signers, effectively diminishing the purported high efficiency. Recent attempts, at best, assume all signers have generated many pre-signatures in an offline stage, which might not be realistic and makes the problems easier. Our robust threshold ECDSA scheme achieves the threshold functionality "for real" – if some participating signers cheated in any stage, the rest can continue as long as any t of them remain. We hope our pursuit of threshold flexibility throughout all protocol phases brings a new perspective to future research. We share some future research directions in Appendix D.

[15]https://github.com/ZenGo-X/multi-party-ecdsa

[16]https://github.com/ZenGo-X/curv, https://github.com/ZenGo-X/class



(a) An error detected during MtA



(b) An error detected during the share revelation phase

Fig. 10: Total Incoming Communication Cost of A Party for An Error Detected during Pre-sign, with $n \in \{5, 10, 15, 20, 30\}$ Parties and Corruption Threshold $n - 1$ for a "Worst-Case" Comparison

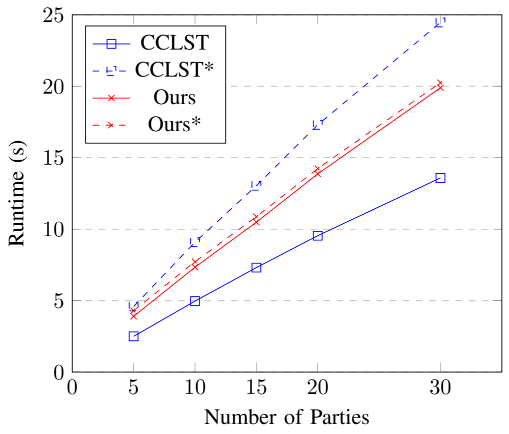


Fig. 11: Total Pre-sign Runtime of A Party with $n \in \{5, 10, 15, 20, 30\}$ and Threshold $n - 1$: The dashed lines indicate the total runtime for correcting an error in Phase 3.

References

[1] D. Abram, A. Nof, C. Orlandi, P. Scholl, and O. Shlomovits, "Low-bandwidth threshold ECDSA via pseudorandom correlation generators," in *S&P*, 2022, pp. 2554–2572.

[2] D. Boneh, B. Bünz, and B. Fisch, "Batching techniques for accumulators with applications to IOPs and stateless blockchains," in *CRYPTO*, 2019, pp. 561–586.

[3] D. Boneh, R. Gennaro, and S. Goldfeder, "Using level-1 homomorphic encryption to improve threshold DSA signatures for bitcoin wallet security," in *LATINCRYPT*, 2017, pp. 352–377.

[4] R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled, "UC non-interactive, proactive, threshold ECDSA with identifiable aborts," in *CCS*, 2020, pp. 1769–1787.

[5] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker, "Two-party ECDSA from hash proof systems and efficient instantiations," in *CRYPTO Part III*. Springer, 2019, pp. 191–221.

[6] ——, "Bandwidth-efficient threshold EC-DSA," in *PKC-II*, 2020, pp. 266–296.

[7] ——, "Bandwidth-efficient threshold EC-DSA revisited: Online/offline extensions, identifiable aborts proactive and adaptive security," *Theor. Comput. Sci.*, vol. 939, pp. 78–104, 2023.

[8] G. Castagnos and F. Laguillaumie, "Linearly homomorphic encryption from DDH," in *CT-RSA*, 2015, pp. 487–505.

[9] R. Cramer, I. Damgård, and J. B. Nielsen, "Multiparty computation from threshold homomorphic encryption," in *EUROCRYPT*, 2001, pp. 280–299.

[10] I. Damgård, T. P. Jakobsen, J. B. Nielsen, J. I. Pagter, and M. B. Østergård, "Fast threshold ECDSA with honest majority," in *SCN*, 2020, pp. 382–400.

[11] J. Doerner, Y. Kondi, E. Lee, and abhi shelat, "Threshold ECDSA from ECDSA assumptions: The multiparty case," in *S&P*, 2019, pp. 1051–1066.

[12] M. Fischlin, "Communication-efficient non-interactive proofs of knowledge with online extractors," in *CRYPTO*, 2005, pp. 152–168.

[13] R. Gennaro and S. Goldfeder, "Fast multiparty threshold ECDSA with fast trustless setup," in *CCS*, 2018, pp. 1179–1194.

[14] R. Gennaro, S. Goldfeder, and A. Narayanan, "Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security," in *ACNS*, 2016, pp. 156–174.

[15] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Robust threshold DSS signatures," in *EUROCRYPT*, 1996, pp. 354–371.

[16] ——, "Secure distributed key generation for discrete-log based cryptosystems," in *EUROCRYPT*, 1999, pp. 295–310.

[17] ——, "Secure applications of Pedersen's distributed key generation protocol," in *CT-RSA*, 2003, pp. 373–390.

[18] A. Gągol, J. Kula, D. Straszak, and M. Świętek, "Threshold ECDSA for decentralized asset custody," IACR Cryptol. ePrint 2020/498, 2020.

[19] J. Groth and V. Shoup, "On the security of ECDSA with additive key derivation and presignatures," in *EUROCRYPT Part I*, 2022, pp. 365–396.

[20] D. Johnson, A. Menezes, and S. A. Vanstone, "The elliptic curve digital signature algorithm (ECDSA)," *Int. J. Inf. Sec.*, vol. 1, no. 1, pp. 36–63, 2001.

[21] Y. Lindell and A. Nof, "Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody," in *CCS*, 2018, pp. 1837–1854.

[22] L. K. L. Ng, S. S. M. Chow, D. P. H. Wong, and A. P. Y. Woo, "LDSP: shopping with cryptocurrency privately and quickly under leadership," in *ICDCS*, 2021, pp. 261–271.

[23] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *CRYPTO*, 1991, pp. 129–140.

[24] M. Pettit, "Efficient threshold-optimal ECDSA," in *CANS*, 2021, pp. 116–135.

[25] D. Pointcheval and J. Stern, "Security arguments for digital signatures and blind signatures," *J. Cryptol.*, vol. 13, no. 3, pp. 361–396, 2000.

[26] V. Shoup and R. Gennaro, "Securing threshold cryptosystems against chosen ciphertext attack," in *EUROCRYPT*, 1998, pp. 1–16.

[27] T. H. Yuen, H. Cui, and X. Xie, "Compact zero-knowledge proofs for threshold ECDSA with trustless setup," in *PKC-I*, 2021, pp. 481–511.

Appendix A
Zero-Knowledge Proof

A. Σ-Protocols and Fiat–Shamir Transform

A Σ-protocol for a statement-witness pair (x, w) in a binary relation $\mathcal{R}$ is a 3-round protocol with PPT algorithms $(\mathsf{P}_1, \mathsf{V}_1, \mathsf{P}_2, \mathsf{V}_2)$ run sequentially.

1) P_1 takes the state τ, outputs the first-round message A.
2) V_1 outputs a random challenge e.
3) P_2 takes (x, w, τ, e) as input and outputs a response z.
4) V_2 takes (x, A, e, z) as input and outputs a bit b.

- *Completeness.* If $(x, w) \in \mathcal{R}$, an honest prover with witness w can always convince an honest verifier that x is true.
- *Special soundness.* There exists a PPT extractor Ext that can extract a witness w' for $(x, w') \in \mathcal{R}$ from two accepting transcripts $(x, A, e, z), (x, A, e' \neq e, z')$ from the prover.
- *Honest-verifier zero-knowledge.* There exists a PPT simulator Sim that takes the statement x and random challenge e as input, and outputs an accepting transcript (x, A, e, z) indistinguishable from a real one to the verifier.

Via Fiat–Shamir heuristic, a Σ-protocol can be compiled into a NIZK proof $(\mathsf{Prove}, \mathsf{Verify})$. In Prove, the prover invokes P_1 that outputs A, and queries (x, A) to the random oracle $\mathsf{H}_{\mathsf{NIZK}}$ to obtain a random challenge e. The prover then invokes P_2 with (x, w, τ, e) to compute z. Verify checks if e equals $\mathsf{H}_{\mathsf{NIZK}}(x, A)$ and invokes V_2 to check the proof (x, A, e, z).

Following Canetti *et al.* [4], we transform a Σ-protocol into NIZK by having the prover first commit to the first-round message via a Com algorithm. The prover with the fixed first-round message can then be rewinded for knowledge extraction.

- $\mathsf{ZKP}_{\mathcal{R}}(\tau).\mathsf{Com} \to A$: Output $A \leftarrow \mathsf{P}_1(\tau)$.
- $\mathsf{ZKP}_{\mathcal{R}}.\mathsf{Prove}(x; w, \tau) \to \pi$: If τ is an empty string, sample $\tau \leftarrow\!\$\ \mathbb{D}$. Run $A \leftarrow \mathsf{P}_1(\tau)$, compute $e := \mathsf{H}_{\mathsf{NIZK}}(A, x)$, run $z \leftarrow \mathsf{P}_2(x, w, \tau, e)$, and output $\pi := (A, e, z)$.
- $\mathsf{ZKP}_{\mathcal{R}}.\mathsf{Verify}(x, \pi) \to 0/1$: Parse $\pi = (A, e, z)$, output 1 if $\mathsf{V}_2(x, A, e, z) = 1$ and $e = \mathsf{H}_{\mathsf{NIZK}}(A, x)$; 0, otherwise.

We also consider an extended 5-round variant $(\mathsf{P}_1, \mathsf{V}_1, \mathsf{P}_2, \mathsf{V}_2, \mathsf{P}_3, \mathsf{V}_3)$ of the Σ-protocol. V_2 not only verifies the response from P_2, but also picks another challenge for P_3. P_3 then generates a response for V_3 to verify.

- V_2 takes in (x, A, e, z) and outputs a random challenge e^*.
- P_3 takes (x, w, τ, e^*) as input and outputs a response z^*.
- V_3 takes (x, A, e^*, z^*) as input and outputs a bit b.

We transform the 5-round protocol into NIZK in the same vein; the first and second challenges are obtained by querying the random oracle on the input of the entire transcript at that point, *i.e.*, (A, x) and (A, x, e, z), respectively. The soundness argument of this 5-round transformation follows from the existing argument [25], which shows that if a constant-round Σ-protocol is sound, then NIZK from Fiat–Shamir transformation is sound. Furthermore, a hash-to-prime function [2] can be used when a prime challenge is required.

B. Σ-Protocols for some NP-Relations

Our scheme involves two types of relations categorized by the mathematical structure involved: EC-group-based ($\mathcal{R}_{\mathsf{DL\text{-}PC}}$ and $\mathcal{R}_{\mathsf{DL\text{-}2PC}}$) and class-group-based ($\mathcal{R}_{\mathsf{key}}$, $\mathcal{R}_{\mathsf{Enc\text{-}PC}}$, and $\mathcal{R}_{\mathsf{Dec}}$). Their instantiations are given in Figures 12, 13, 14.

$$\begin{array}{lcl}
\textbf{Prover}((\mathsf{PC}, g_0, Q), (m, r)) & & \textbf{Verifier}(\mathsf{PC}, g_0, Q) \\
\hline
r_1, r_2 \leftarrow\!\$\ \mathbb{F}_q, T_1 := g^{r_1}h^{r_2} & & \\
T_2 := g_0^{r_1} & \xrightarrow{T_1, T_2} & \\
 & \xleftarrow{k} & k \leftarrow\!\$\ \mathbb{F}_q \\
u_1 := r_1 + km, u_2 := r_2 + kr & \xrightarrow{u_1, u_2} & \textbf{check } g^{u_1}h^{u_2} \overset{?}{=} T_1\mathsf{PC}^k, \\
 & & g_0^{u_1} \overset{?}{=} T_2 Q^k
\end{array}$$

Fig. 12: Σ-Protocol for $\mathcal{R}_{\mathsf{DL\text{-}PC}}$

$$\begin{array}{lcl}
\textbf{Prover}((\mathsf{PC}, \ldots, R), (k, k', \mu)) & & \textbf{Verifier}(\mathsf{PC}, \ldots, R) \\
\hline
r_1, r_2, r_3 \leftarrow\!\$\ \mathbb{Z}_q, T_1 := g^{r_1}h^{r_2} & & \\
T_2 := X^{r_1}(1/g)^{r_3}, T_3 := R^{r_3} & \xrightarrow{T_1, T_2, T_3} & \\
u_1 := r_1 + ck, u_2 := r_2 + ck' & \xleftarrow{c} & c \leftarrow\!\$\ \mathbb{Z}_q \\
u_3 := r_3 + c\mu & \xrightarrow{u_1, u_2, u_3} & \textbf{check } g^{u_1}h^{u_2} \overset{?}{=} T_1\mathsf{PC}^c, \\
 & & X^{u_1}(1/g)^{u_3} \overset{?}{=} T_2\mathcal{N}^c, \\
 & & R^{u_3} \overset{?}{=} T_3 R^c
\end{array}$$

Fig. 13: Σ-Protocol for $\mathcal{R}_{\mathsf{DL\text{-}2PC}}$

The EC-group-based relations can be proved by the combined use of Schnorr protocol and Okamoto protocol. The constructions are standard, so we omit the security analysis.

For class-group-based relations, we adapt the 5-round Σ-protocol [27] by Yuen *et al.* More concretely, we use the protocol [27, Algorithm 2] for $\mathcal{R}_{\mathsf{key}}$ and modify [27, Algorithm 6] for $\mathcal{R}_{\mathsf{Enc\text{-}PC}}$ and $\mathcal{R}_{\mathsf{Dec}}$, respectively. The special soundness property relies on the adaptive root subgroup assumption[17] in the generic group model. We follow the parameter [27] such that the soundness error and statistical distance are 2^{-80}.

The security analysis of Σ-protocol for $\mathcal{R}_{\mathsf{Enc\text{-}PC}}$ follows the 5-round protocol [27, Algorithm 6] with some minor changes for the newly added term $r \in \mathbb{Z}_q$. To argue special soundness, we construct a new extractor from the old extractor of the 5-round protocol. Given two accepted transcripts, the new extractor extracts the whole witness (m, r, ρ). In more detail, (m, ρ) is extracted via the old extractor; given the transcripts including terms (k, u_3, k', u_3'), the new term r is computed as $(u_3 - u_3')/(k - k') \in \mathbb{Z}_q$. We argue it is zero-knowledge via a similar method. We construct a new simulator that simulates the whole transcript $(T, T_3, k, u, u_3, p, u')$[18]. The term (T, k, u, p, u') of the original protocol is simulated via the old simulator; the new term u_3 is randomly sampled, and T_3 is computed as $g^{u_2}h^{u_3}/\mathsf{PC}^k \in \mathbb{G}$.

The proof for CL decryption is simplified by revealing m

[17]Informally, for an adversary-chosen element w and a prime challenge ℓ, it is hard to compute the root u, *i.e.*, $u^\ell = w$. See [27] for more detail.

[18]We write the commitment, first response, and second response of the original protocol as $T := (T_1, T_2)$, $u := (u_1, D_{q,1}, D_{q,2}, e_q)$, and $u' := (D_{p,1}, D_{p,2}, e_p)$, respectively.

$$\begin{array}{lcl}
\textbf{Prover}((\mathsf{PC},\mathsf{ek},(c_1,c_2)),(m,r,\rho)) & & \textbf{Verifier}(\mathsf{PC},\mathsf{ek},(c_1,c_2)) \\
\hline
r_1 \leftarrow\!\!\$\ \mathbb{Z}_q, r_2 \leftarrow\!\!\$\ [-B,B], r_3 \leftarrow\!\!\$\ \mathbb{Z}_q & & \\
T_1 := \mathfrak{g}_q^{r_2}, T_2 := \mathfrak{f}^{r_1}\mathsf{ek}^{r_2}, T_3 := g^{r_1}h^{r_3} & \xrightarrow{T_1,T_2,T_3} & \\
u_1 := r_1 + km, u_2 := r_2 + k\rho, u_3 := r_3 + kr & \xleftarrow{k} & k \leftarrow\!\!\$\ \mathbb{Z}_q \\
\textbf{pick } d_q \in \mathbb{Z}, e_q \in \mathbb{Z}_q \text{ s.t. } u_2 = d_q q + e_q & & \\
D_{q,1} := \mathfrak{g}_q^{d_q}, D_{q,2} := \mathsf{ek}^{d_q} & \xrightarrow{u_1,u_3,D_{q,1},D_{q,2},e_q} & \textbf{check } e_q \in \mathbb{Z}_q, D_{q,1}^q\mathfrak{g}_q^{e_q} \overset{?}{=} T_1c_1^k, \\
 & & \mathfrak{f}^{u_1}D_{q,2}^q\mathsf{ek}^{e_q} \overset{?}{=} T_2c_2^k, g^{u_1}h^{u_3} \overset{?}{=} T_3\mathsf{PC}^k \\
\textbf{pick } d_p \in \mathbb{Z}, e_p \in \mathbb{Z}_p \text{ s.t. } u_2 = d_p p + e_p & \xleftarrow{p} & p \leftarrow\!\!\$\ \mathsf{Primes}(\lambda) \\
D_{p,1} := \mathfrak{g}_q^{d_p}, D_{p,2} := \mathsf{ek}^{d_p} & \xrightarrow{D_{p,1},D_{p,2},e_p} & \textbf{check } e_p \in \mathbb{Z}_p, D_{p,1}^p\mathfrak{g}_q^{e_p} \overset{?}{=} T_1c_1^k, \mathfrak{f}^{u_1}D_{p,2}^p\mathsf{ek}^{e_p} \overset{?}{=} T_2c_2^k
\end{array}$$

Fig. 14: Σ-Protocol for $\mathcal{R}_{\mathsf{Enc\text{-}PC}}$

from decrypting (c_1, c_2), and proving about $c_2' = c_2/\mathfrak{f}^m$:

$$\mathcal{R}_{\mathsf{Dec}} = \{((\mathsf{ek},(c_1,c_2')),\mathsf{dk})\colon \mathsf{dk} \in [0,S] \wedge c_2' = c_1^{\mathsf{dk}} \wedge \mathsf{ek} = \mathfrak{g}_q^{\mathsf{dk}}\}.$$

Recall that a well-formed ciphertext satisfies the following:

$$\mathcal{R}_{\mathsf{Enc}} = \{((\mathsf{ek},c),(m,\rho)) : \rho \in [0,S] \wedge c_1 = \mathfrak{g}_q^{\rho} \wedge c_2 = \mathfrak{f}^m\mathsf{ek}^{\rho}\}.$$

When we set $m = 0$ and $\rho = \mathsf{dk}$, $\mathcal{R}_{\mathsf{Dec}}$ becomes $\mathcal{R}_{\mathsf{Enc}}$. ZKP for $\mathcal{R}_{\mathsf{Enc\text{-}PC}}$ can be used to prove $\mathcal{R}_{\mathsf{Dec}}$ by removing m from the witness, *i.e.*, $\mathsf{ZKP}_{\mathsf{Dec}}((\mathsf{ek},(c_1,c_2'));\mathsf{dk}) = \mathsf{ZKP}_{\mathsf{Enc\text{-}PC}}((\mathsf{ek},(c_1,c_2'));(0,\mathsf{dk}))$ and setting $(\mathsf{PC}, r) = (1, 0)$.

Appendix B
Analysis of Distributed Randomness Generation

A. Correctness and Self-healing

Theorem 1: If NIZK is sound and zero-knowledge, VSS satisfies correctness, robustness, and unconditional secrecy, and CLE is indistinguishable against adaptive chosen-plaintext attacks (IND-CPA-secure), then our DRG is correct and self-healing (with $<t$ corrupted parties and an honest majority).

Proof: Uniform Distribution. After the generation phase, x is fixed as $\sum_{j\in\mathcal{Q}_x}\chi_j$. Its distribution can only be influenced during the generation phase. The view of the adversary (taking the role of party j) regarding the output of party i includes:

1) $(F_{\chi_i}, \chi_{ij}, \chi'_{ij})$, which is independent of the secret χ_i of the honest party i due to the secrecy of VSS;
2) π_{χ_i} and c_{χ_i} leak negligible information about χ_i since NIZK is zero-knowledge and CLE is IND-CPA-secure.

The adversary can only corrupt less than t parties. It cannot learn any partial information about the secret χ_i or cancel the random contributions of honest parties. So, $x = \chi_i + (\sum_{j\in\mathcal{Q}_x\setminus\{i\}}\chi_j)$ is uniformly distributed.

Correct Public Shares. Gen and GenVf run VSS to (threshold) share and verify each party's contribution χ_i. Any inconsistent shares/outputs of verification materials, commitments, and ciphertexts would be detected (by the soundness of ZKP and security of VSS) and removed. Thus, the verification materials F_{χ_i} and shares $\{\chi_{ij}\}_{i,j}$ are correct.

In CombVf and CombVf, $\mathcal{R}_{\mathsf{Enc\text{-}PC}}$ ensures that x_i satisfies the combined Pedersen commitment (which implies $x = \sum_{j\in\mathcal{Q}_x}\chi_j$ and $x_i = \sum_{j\in\mathcal{Q}_x}\chi_{ji}$) and is correctly encrypted. (RevealExp, ExpVf) proves the above share x_i in exponent is also committed, which is ensured by $\mathcal{R}_{\mathsf{DL\text{-}PC}}$.

Robust Reconstruction. For correct contribution $\{x_j\}_{j\in\mathcal{P}}$, any subset $\mathcal{P}' \subseteq \mathcal{P}$ ($|\mathcal{P}'| \geq t$) of the remaining parties can robustly reconstruct $x := \sum_{j\in\mathcal{Q}_x}\chi_j$ as follows:

$$\begin{aligned}\sum_{i\in\mathcal{P}'} L_{i,\mathcal{P}'}x_i &= \sum_{i\in\mathcal{P}'} L_{i,\mathcal{P}'}\left(\sum_{j\in\mathcal{Q}_x}\chi_{ji}\right) \\ &= \sum_{j\in\mathcal{Q}_x}\left(\sum_{i\in\mathcal{P}'} L_{i,\mathcal{P}'}\chi_{ji}\right) = \sum_{j\in\mathcal{Q}_x}\chi_j.\end{aligned}$$

■

By the robust reconstruction, we know that a sufficient number of ($\geq t$) parties can guarantee protocol output.

B. Secrecy

In the real protocol, each party generates a CLE key pair for committing to the secret values. Our security analysis expects the simulator obtains the decryption keys of corrupted parties for an (online) extractor, which reduces rewinding and to avoid common issues [26] in the multiparty or concurrent setting.

Theorem 2: If NIZK is zero-knowledge, VSS has unconditional secrecy, and CLE is IND-CPA-secure, DRG has secrecy over discrete-logarithm groups assuming random oracle.

Proof: Let $\mathcal{H}$ and $\mathcal{P}$ be the set of honest and all parties, respectively. We outline a simulator $\mathcal{S}_{\mathsf{DRG}}$ taking input X and decryption keys $\{\mathsf{dk}_j\}_{j\in\mathcal{C}}$ to simulate the view for a PPT adversary $\mathcal{A}$ corrupting, w.l.o.g., the parties $\mathcal{C} = \{1,\ldots,t-1\}$ Gen gives shares $\{\chi_{ij},\chi'_{ij}\}_{j\in\mathcal{P}}$, a ciphertext c_{χ_i}, Pedersen commitments F_{χ_i}, and ZKP π_{χ_i} of party i. Comb gives Pedersen commitments PC_{x_i}, ciphertext c_{x_i}, and ZKP π_{x_i} of party i. By the secrecy of VSS, perfect hiding of Pedersen commitments, IND-CPA-security of CLE, and zero-knowledgeness of NIZK, the indistinguishable views of $(\mathsf{Gen}, \mathsf{Comb})$ can be simulated. (GenVf, CombVf and ExpVf are omitted as they only verify.) It remains to show that the simulated view of RevealExp is indistinguishable from a real one.

At a high level, we simulate the share in exponent $\{X_i\}_{i\in\mathcal{H}}$ for a given X as follows. In the real protocol (generation phase), each party $i \in \mathcal{H}$ (controlled by the simulator) picks an initial secret χ_i and sends the VSS shares χ_{ij} of χ_i to other party $j \in \mathcal{P} \setminus \{i\}$. The simulator receives from the adversary their contribution (encryption of χ_j and shares of χ_j from party $j \in \mathcal{C}$). Let $Y_{\mathcal{C}} = g^{\sum_{j\in\mathcal{C}} \chi_j}$ be the lifted product of all contributions from the corrupted parties (with χ_j extracted/decrypted from the CLE ciphertext). Write $Y_{\mathcal{H}} = X/Y_{\mathcal{C}}$. The simulator can compute the adversary's view in RevealExp $(\{X_i, \pi_{X_i}\}_{i\in\mathcal{H}})$ by "sharing" $Y_{\mathcal{H}}$ using shares received from (and sent to) the adversary in the generation phase and by invoking the zero-knowledge simulator to simulate proof for $\mathcal{R}_{\mathsf{DL\text{-}PC}}$ on a random CLE ciphertext. ∎

Appendix C
Security Analysis of Threshold ECDSA

A. Correctness and Self-healing

Like prior threshold ECDSA schemes with optimal threshold [14], [24], we consider security against static corruption.

We say an online/offline (t, n) threshold ECDSA protocol is correct (against $(t-1)$ corrupted parties) if, for $\mathsf{TKeygen}\langle \mathsf{pp}; \ldots; \mathsf{pp}\rangle \rightarrow \langle(\mathsf{vk}, \mathsf{pk}, \mathsf{sk}_1); \ldots; (\mathsf{vk}, \mathsf{pk}, \mathsf{sk}_n)\rangle$, $\mathsf{PreSign}\langle(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_1); \ldots; (\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_{n'})\rangle \rightarrow \langle\psi_1; \ldots; \psi_{n'}\rangle$, and $\mathsf{TSign}\langle(\mathsf{pp}, \mathsf{pk}, \mathsf{vk}, \mathsf{msg}, \mathsf{sk}_1, \psi_1); \ldots)\rangle \rightarrow \sigma$,

$$\Pr[\mathsf{Verify}(\mathsf{vk}, \sigma, \mathsf{msg}) = 1] \geq 1 - \mathsf{negl}(\lambda).$$

To show the correctness, we show that the (shares of) the ingredient of the signature σ can be verified, and cheaters can be identified (and removed). Since our protocol works over (t, n) sharing back-to-back, self-healing can be achieved by removing cheaters' contributions.

1) TKeygen: TKeygen is correct if the output $(\mathsf{vk}, \mathsf{pk} := \{\mathsf{ek}_j, X_j\}_{j\in\mathcal{U}}, \mathsf{sk}_i := (\mathsf{dk}_i, x_i))$ of an honest party satisfies the following requirements for a secret x (that can be reconstructed by some subset of size $\geq t$ of $\mathcal{U}$):

1) $\{\mathsf{ek}_j\}_{j\in\mathcal{U}}$ contains the CLE public key of each party j;
2) $\mathsf{vk} = g^x$, $\{X_j = g^{x_j}\}_{j\in\mathcal{U}}, x_i$ where x_j is party j's share of x (from DRG).

Lemma 2: If DRG is correct (self-healing) and Σ_{key} is sound, the TKeygen protocol in Figures 3 and 4 outputs a valid tuple $(\mathsf{vk}, \mathsf{pk}, \{\mathsf{sk}_j\}_{j\in\mathcal{U}})$ for the set $\mathcal{U}$ of honest participants.

Proof: TKeygen invokes DRG to generate $\mathsf{vk} = g^x$ and shares of x: $\{x_j, X_j = g^{x_j}\}_{j\in\mathcal{U}}$. By the robustness of DRG, the honest parties can detect and remove inconsistent values. Moreover, ZKP ensures CLE key-pair $(\mathsf{ek}_j, \mathsf{dk}_j) \in \mathcal{R}_{\mathsf{key}}$. ∎

2) PreSign: PreSign is correct if the outputs to an honest party i, $V_i = (k'_i, \{\mathsf{PC}_{k_j}\}_{j\in\mathcal{P}}, \{R_j\}_{j\in\mathcal{P}}, \{\{\mathcal{N}_{j\ell}\}_{\ell\in\mathcal{P}\setminus\{j\}}\}_{j\in\mathcal{P}})$ and $\varphi_i = (R, k_i, \{(\mu_{ij}, \nu_{ji})\}_{j\in\mathcal{P}\setminus\{i\}})$, satisfy:

1) $R = g^{1/k}$, $\{R_j = g^{k_i/k}\}_{j\in\mathcal{P}}$;
2) k_i is party i's (threshold) share of k;
3) $\{(\mu_{ij}, \nu_{ji})\}_{j\in\mathcal{P}\setminus\{i\}}$ satisfies $\mu_{ij} + \nu_{ij} = k_i x_j$;
4) $\{\{\mathcal{N}_{j\ell} = g^{\nu_{j\ell}}\}_{\ell\in\mathcal{P}\setminus\{j\}}\}_{j\in\mathcal{P}}$;
5) $\{\mathsf{PC}_{k_j} = g^{k_j} h^{k'_j}\}_{j\in\mathcal{P}}$;

Lemma 3: If DRG and CLE are correct, and $\mathsf{NIZK}_{\mathsf{DL\text{-}PC}}$ is sound, the PreSign protocol in Figures 5 and 6 outputs a valid pre-signature $\{\varphi_i, V_i\}$ for the set of honest participants.

Proof: We show that relations are built in each phase to make the requirements listed above hold.

Phase 1: Parties run two instances of DRG for k and γ. By DRG correctness (and robustness), the following relations hold, and thus items (2) and (5) are satisfied.

- $\{k_j, \gamma_j\}_{j\in\mathcal{P}}$ are shares of k, γ; RevealExp is invoked for γ, *e.g.*, outputting $\{\Gamma_j\}_{j\in\mathcal{P}}$
- $\{c_{k_j}, \mathsf{PC}_{k_j}\}_{j\in\mathcal{P}}$ in pub_{k_j} are encrypting and committing k_j.

Phase 2: We have correct shares of γ in exponent $\{\Gamma_j = g^{\gamma_j}\}_{j\in\mathcal{P}}$ by the robustness of DRG. Each party i runs MtAwc with party $j \in \mathcal{P}\setminus\{i\}$ to compute additive shares of $k_j\gamma_i, k_j x_i$. By the MtAwc protocol, $\{\alpha_{ij}, \beta_{ij}, \mu_{ij}, \nu_{ij}\}_{i\neq j}$ are additive shares of $k_i\gamma_j, k_i x_j$, *i.e.* $\alpha_{ij} + \beta_{ij} = k_i\gamma_j$ and $\mu_{ij} + \nu_{ij} = k_i x_j$; $\{\mathcal{B}_{ji}, \mathcal{N}_{ji}\}_{i\neq j}$ are correct additive shares in exponent such that $\mathcal{B}_{ji} = g^{\beta_{ji}}$ and $\mathcal{N}_{ji} = g^{\nu_{ji}}$. The decrypted shares can be checked by $g^{\alpha_{ji}}\mathcal{B}_{ji} \stackrel{?}{=} \Gamma_i^{k_j}$ and $g^{\mu_{ji}}\mathcal{N}_{ji} \stackrel{?}{=} X_i^{k_j}$, where the variables in capital letters are broadcasted. A party can complain the sender by broadcasting proof of decryption against the broadcasted ciphertext from MtAwc, and remove the cheater. Thus, conditions (3) and (4) are satisfied.

Phase 3: Each party i reveals shares $\{\delta_{ij}\}_{j\in\mathcal{P}}$. Recall the matrix notation in Section V-B3, $g^{\mathbf{T}}$ can be computed using $\{\mathcal{B}_{ij}\}_{i,j}$ broadcasted in Phase 2 (all values are "trusted" if there is no complaint); $\mathbf{S}[i, j]$ is the unmasked shares broadcasted in Phase 3, *i.e.* $\mathbf{S}[i, j] = \delta_{ij} - \theta_{ij}$. Also note that $\{\Gamma_i\}$ revealed in RevealExp during Phase 2 is used to compute $\Gamma = g^\gamma$ (ensured by the robustness of DRG), and $D_i = \Gamma^{k_i}$ (ensured by the soundness of NIZK for $\mathcal{R}_{\mathsf{DL\text{-}PC}}$), which correspond to exponents in $g^{\mathbf{S}\cdot\vec{\ell}+\mathbf{T}\cdot\vec{\ell}}$. Hence, any party can check:

$$g^{\delta_{i,\mathcal{P}}} \prod_{j\in\mathcal{P}\setminus\{i\}} (\mathcal{B}_{ij}/\mathcal{B}_{ji})^{L_{j,\mathcal{P}}} \stackrel{?}{=} D_i,$$

where $\delta_{i,\mathcal{P}} = (L_{1,n}(\alpha_{i1} + \beta_{1i}) + \cdots + L_{i,n}k_i\gamma_i + \cdots + L_{n,n}(\alpha_{in} + \beta_{ni}))$, and mark party i as a cheater if the equation does not hold. Thus, the following conditions hold:

- $\{D_j\}_{j\in\mathcal{P}}$ are correct shares in exponent to the base element g^γ, *i.e.*, $\{D_j = (g^\gamma)^{k_j}\}_{j\in\mathcal{P}}$, ensured by NIZK for $\mathcal{R}_{\mathsf{DL\text{-}PC}}$.
- $\{\delta_{ij}\}_{i,j\in\mathcal{P}}$ can robustly reconstruct $k\gamma$, *i.e.*, for any subset $\mathcal{P}' \subseteq \mathcal{P}$ of size $\geq t$, we have $\sum_{i\in\mathcal{P}'} L_{i,\mathcal{P}'}\delta_{i,\mathcal{P}'} = k\gamma$, where $\delta_{i,\mathcal{P}'} = \sum_{j\in\mathcal{P}'} L_{j,\mathcal{P}'}\delta_{ij}$. If the "first-layer" reconstruction $\delta_{i,\mathcal{P}'}$ of broadcasted shares $\{\delta_{ij}\}_{j\in\mathcal{P}}$ for $i \in \mathcal{P}'$ satisfies Equation (2), we have $\sum_{i\in\mathcal{P}'} L_{i,\mathcal{P}'}\delta_{i,\mathcal{P}'} = k\gamma$:

$$\prod_{i\in\mathcal{P}'} \left(g^{\delta_{i,\mathcal{P}'}} \prod_{j\in\mathcal{P}'\setminus\{i\}} (\mathcal{B}_{ij}/\mathcal{B}_{ji})^{L_{j,\mathcal{P}'}} \right)^{L_{i,\mathcal{P}'}} = \prod_{i\in\mathcal{P}'} D_i^{L_{i,\mathcal{P}'}}$$
$$\prod_{i\in\mathcal{P}'} (g^{\delta_{i,\mathcal{P}'}})^{L_{i,\mathcal{P}'}} = (g^\gamma)^{\sum_{j\in\mathcal{P}'} L_{j,\mathcal{P}'}k_j}$$
$$g^{\sum_{i\in\mathcal{P}'} L_{i,\mathcal{P}'}\delta_{i,\mathcal{P}'}} = g^{k\gamma}.$$

Thus, condition (1) is achieved. In the output phase, each party computes $R := \Gamma^{1/\delta} = g^{1/k}$ and $\{R_j := D_j^{1/\delta} =$

$g^{k_i/k}\}_{j\in\mathcal{P}}$, and outputs $\varphi_i := (R, k_i, \{\mu_{ij}, \nu_{ji}\}_{j\in\mathcal{P}\setminus\{i\}})$ and $V_i := (k_i', \{\mathsf{PC}_{k_j}\}_{j\in\mathcal{P}}, \{R_j\}_{j\in\mathcal{P}}, \{\{\mathcal{N}_{j\ell}\}_{\ell\in\mathcal{P}\setminus\{j\}}\}_{j\in\mathcal{P}})$, which satisfy conditions (1) to (5). ■

3) TSign*:* We say TSign is correct if it outputs $\sigma := (r, s)$ such that $g^m X^r = R^s$, where $m = H(\mathsf{msg})$, r is the x-projection of the EC-point R, and X is the verification key.

Lemma 4: For any honestly generated pre-signature from PreSign, the TSign protocol in Figures 7, 8, and 9 outputs a valid signature for the set of honest participants.

Proof: Each party i generates threshold signature $\{s_{ij}\}_{j\in\mathcal{P}^*}$ and verification material $\mathbb{V}_i$, including $\{\mathcal{M}_{ij}\}_{j\in\mathcal{P}^*}$.

- $\{\mathcal{M}_{ij}\}_{j\in\mathcal{P}^*}$ are correct additive shares in exponent, *i.e.*, $\{\mathcal{M}_{ij} = R^{\mu_{ij}}\}_{j\in\mathcal{P}^*}$. This relation is ensured by the soundness of $\mathsf{NIZK}_{\mathsf{DL\text{-}2PC}}$ against the commitment PC_{k_i}.
- $\{s_{ij}\}_{i,j\in\mathcal{P}^*}$ can robustly reconstruct $km + rkx$, *i.e.*, for any subset $\mathcal{P}^{*\prime} \subseteq \mathcal{P}^*$ ($|\mathcal{P}^{*\prime}| \geq t$), we have $km + rkx = \sum_{i\in\mathcal{P}^{*\prime}} L_{i,\mathcal{P}^{*\prime}} s_{i,\mathcal{P}^{*\prime}}$, where $s_{i,\mathcal{P}^{*\prime}} = \sum_{j\in\mathcal{P}^{*\prime}} L_{j,\mathcal{P}^{*\prime}} s_{ij}$.
  In a similar vein to the argument for $\{\delta_{ij}\}_{i,j}$ of PreSign Phase 3: if the "first-layer" reconstruction $s_{i,\mathcal{P}^{*\prime}}$ of the threshold signature $\{s_{ij}\}_{j\in\mathcal{P}^*}$ satisfies Equation 3:
  $$R^{s_{i,\mathcal{P}^{*\prime}}} \prod_{j\in\mathcal{P}^{*\prime}\setminus\{i\}} (\mathcal{M}_{ji}/\mathcal{M}_{ij})^{rL_{j,\mathcal{P}^{*\prime}}} = R_i^m X_i^r,$$
  we have:
  $$\begin{aligned}&\prod_{i\in\mathcal{P}^{*\prime}} (R_i^m X_i^r)^{L_{i,\mathcal{P}^{*\prime}}}\\ &= \prod_{i\in\mathcal{P}^{*\prime}} \left(R^{s_{i,\mathcal{P}^{*\prime}}} \prod_{j\in\mathcal{P}^{*\prime}\setminus\{i\}} (\mathcal{M}_{ji}/\mathcal{M}_{ij})^{rL_{j,\mathcal{P}^{*\prime}}} \right)^{L_{i,\mathcal{P}^{*\prime}}}\\ &= \prod_{i\in\mathcal{P}^{*\prime}} (R^{s_{i,\mathcal{P}^{*\prime}}})^{L_{i,\mathcal{P}^{*\prime}}}\\ &= R^{\sum_{i\in\mathcal{P}^{*\prime}} L_{i,\mathcal{P}^{*\prime}} s_{i,\mathcal{P}^{*\prime}}}.\end{aligned}$$
  Note that $\prod_{i\in\mathcal{P}^{*\prime}} (R_i^m X_i^r)^{L_{i,\mathcal{P}^{*\prime}}} = R^{L_{i,\mathcal{P}^{*\prime}}(k_i m + rkx_i)} = R^{km+rkx}$. Therefore, we have $\sum_{i\in\mathcal{P}^{*\prime}} L_{i,\mathcal{P}^{*\prime}} s_{i,\mathcal{P}^{*\prime}} = km + rkx$, and $\{s_{i,\mathcal{P}'}\}_{i\in\mathcal{P}^{*\prime}}$ can reconstruct $s = km + rkx$.
  In more detail, shares $\{s_{ij}\}_{j\in\mathcal{P}^*}$ defined by $\{k_i m_{ii} + r(k_i x_i)\} \cup \{k_i m_{ij} + r(\mu_{ij} + \nu_{ji})\}_{j\in\mathcal{P}^*\setminus\{i\}}\}$ pass the above verification. To see, for any subset $\mathcal{P}^{*\prime} \subseteq \mathcal{P}^*$, $k_i m = \sum_{j\in\mathcal{P}^{*\prime}} L_{j,\mathcal{P}^{*\prime}} k_i m_{ij}$ and $m = \sum_{j\in\mathcal{P}^{*\prime}} L_{j,\mathcal{P}^{*\prime}} m_{ij}$, $s_{i,\mathcal{P}^{*\prime}} = k_i m + r(L_{i,\mathcal{P}^{*\prime}} k_i x_i + \sum_{j\in\mathcal{P}^{*\prime}\setminus\{i\}} L_{j,\mathcal{P}^{*\prime}}(\mu_{ij} + \nu_{ji}))$.
  This gives us the following evaluation:
  $$\begin{aligned}&R^{s_{i,\mathcal{P}^{*\prime}}} \textstyle\prod_{j\in\mathcal{P}^{*\prime}\setminus\{i\}} (\mathcal{M}_{ji} / \mathcal{M}_{ij})^{rL_{j,\mathcal{P}^{*\prime}}}\\ =\, &R^{k_i m} R^{L_{i,\mathcal{P}^{*\prime}} k_i x_i} \textstyle\prod_{j\in\mathcal{P}^{*\prime}\setminus\{i\}} (R^{\nu_{ji}} \cdot R^{\mu_{ij}})^{rL_{j,\mathcal{P}^{*\prime}}}\\ &\textstyle\prod_{j\in\mathcal{P}^{*\prime}\setminus\{i\}} (\mathcal{M}_{ji} / \mathcal{M}_{ij})^{rL_{j,\mathcal{P}^{*\prime}}}\\ =\, &R_i^m R^{L_{i,\mathcal{P}^{*\prime}} k_i x_i} \textstyle\prod_{j\in\mathcal{P}^{*\prime}\setminus\{i\}} (R^{\nu_{ji}} \mathcal{M}_{ji} \cdot R^{\mu_{ij}} / \mathcal{M}_{ij})^{rL_{j,\mathcal{P}^{*\prime}}}\\ =\, &R_i^m R^{L_{i,\mathcal{P}^{*\prime}} k_i x_i} \textstyle\prod_{j\in\mathcal{P}^{*\prime}\setminus\{i\}} (R^{k_j x_i} \cdot 1)^{rL_{j,\mathcal{P}^{*\prime}}}\\ =\, &R_i^m X_i^r.\end{aligned}$$

The above holds for any subset $\mathcal{P}^{*\prime}$ of size $\geq t$ for reconstructing $s = km + rkx$, so TSign is self-healing and correct. ■

B. Unforgeability of Threshold ECDSA

If an adversary $\mathcal{A}$ breaks the unforgeability of our protocol with non-negligible probability, we show how to build a forger $\mathcal{F}$ that breaks the enhanced unforgeability of ECDSA with non-negligible probability. $\mathcal{F}$ comprises three simulators ($\mathcal{S}_{\mathsf{TKeygen}}, \mathcal{S}_{\mathsf{PreSign}}, \mathcal{S}_{\mathsf{TSign}}$). These simulators generate views indistinguishable from (TKeygen, PreSign, TSign) when interacting with $\mathcal{A}$ in the real execution of the unforgeability game.

Let $\mathcal{C}, \mathcal{H}$ be the corrupted set and the honest set, respectively. Let $\mathcal{U}$, $\mathcal{P}$, $\mathcal{P}^*$ be the set of parties who participate and are not recognized as a cheater in TKeygen, PreSign, TSign, respectively. The set $\mathcal{U}$, $\mathcal{P}$, $\mathcal{P}^*$ are updated whenever a cheater is identified. $\mathcal{F}$ handles all honest parties to maintain consistency. We assume w.l.o.g. that $\mathcal{F}$ takes the role of party i.

1) Simulation of Key Generation Protocol: $\mathcal{F}$ aims to mold the verification key vk in $\mathcal{S}_{\mathsf{TKeygen}}$ as X, which is received from the enhanced existential unforgability game of ECDSA. Recall that TKeygen consists of KGen of CL encryption (Phase 1) and DRG protocol (Phase 2). $\mathcal{F}$ first uses the knowledge extractor to extract decryption keys $\{\mathsf{dk}_j\}_{j\in\mathcal{C}}$. With $\{\mathsf{dk}_j\}_{j\in\mathcal{C}}$, $\mathcal{F}$ then invokes the simulator $\mathcal{S}_{\mathsf{DRG}}$ of DRG, which simulates an indistinguishable view of DRG such that the protocol uses X as the public signing key. For the simulated view from $\mathcal{S}_{\mathsf{DRG}}$, by Lemma 2, the view of $\mathcal{S}_{\mathsf{DRG}}$ and DRG are indistinguishable. Thus, $\mathcal{S}_{\mathsf{TKeygen}}$ and TKeygen are indistinguishable.

2) Simulation of Pre-signing Protocol: $\mathcal{F}$ aims to mold the nonce in $\mathcal{S}_{\mathsf{PreSign}}$ as R, which is received from $\mathcal{O}^{\mathsf{Rand}}$. $\mathcal{S}_{\mathsf{PreSign}}$ is separated into four phases as PreSign protocol. The output phase only involves local computation using existing ingredients, and no simulation is required. It follows the output phase of $\mathcal{S}_{\mathsf{PreSign}}$ and is omitted here. For brevity, we also omitted the ZKP that can be simulated by the ZKP simulator.

In Phase 1, $\mathcal{F}$ extracts $\{k_j, \gamma_j\}_{j\in\mathcal{P}}$ by decrypting the encrypted shares in $(\mathsf{pub}_{k_i}, \mathsf{pub}_{\gamma_j})$. From $\{k_j, \gamma_j\}_{j\in\mathcal{P}}$, $\mathcal{F}$ computes $k := \sum_{i\in\mathcal{P}} L_{i,\mathcal{P}} k_i$, $\gamma := \sum_{i\in\mathcal{P}} L_{i,\mathcal{P}} \gamma_i$, and $\delta := k\gamma$.

In Phase 2, with R from $\mathcal{O}^{\mathsf{Rand}}$, $\mathcal{F}$ simulates $\Gamma := R^\delta$ and shares Γ_i of Γ (note that $R \neq g^{1/k}$, $\Gamma \neq g^\gamma$ in the simulation).

$\mathcal{F}$ plays as initiator i for respondent j (in MtAwc). Without knowing the discrete logarithm of simulated values Γ_i (X_i), $\mathcal{F}$ randomly samples the additive shares α_{ji} (μ_{ji}) for respondent j. $\mathcal{F}$ then simulates $\mathcal{B}_{ji} := {\Gamma_i}^{k_j}/g^{\alpha_{ji}}$ ($\mathcal{N}_{ji} := {X_i}^{k_j}/g^{\mu_{ji}}$) and ciphertext $c_{\alpha_{ji}} \leftarrow \mathsf{Enc}(\mathsf{ek}_j, \alpha_{ji})$ ($c_{\mu_{ji}} \leftarrow \mathsf{Enc}(\mathsf{ek}_j, \mu_{ji})$).

$\mathcal{F}$ plays as respondent i for initiator j and decrypts the ciphertext $c_{\alpha_{ij}}$ ($c_{\mu_{ij}}$) with decryption key dk_i to obtain α_{ij} (μ_{ij}). In addition, since $\mathcal{F}$ knows k_i, it can compute shares $\beta_{ij} := k_i\gamma_j - \alpha_{ij}$ ($\nu_{ij} := k_i x_j - \mu_{ij}$) of party j.

Following the existing proof strategy of [6], [7] for MtAwc, the simulated view is indistinguishable from the view of the real protocol in Phase 2 of PreSign by the indistinguishability argument. Let k' be the randomness used to generate the nonce R received from $\mathcal{O}^{\mathsf{Rand}}$, and we define k_i' as the shares of the randomness k'. Γ_i is simulated as shares of $\Gamma = R^\delta$. Let the discrete logarithm of Γ_i be γ_i', and $\gamma' := L_{i,\mathcal{P}}\gamma_i' + \sum_{j\in\mathcal{C}} L_{j,\mathcal{P}}\gamma_j$. Revealing Γ_i (while requiring $\delta = k'\gamma'$) implicitly uses k_i' in the ciphertext c_{k_i}. It has been shown that in the presence of materials from MtAwc, no PPT adversary can distinguish k_i' from k_i with an advantage greater than $2\epsilon_{\mathsf{HSM}} + 3/q + 4\epsilon_s$, where CL encryption is ϵ_s-smooth, and the hard subgroup membership problem is ϵ_{HSM}-hard. We refer the readers to arguments in the literature [6], [7] for detail.

In Phase 3, $\mathcal{F}$ computes $\{\delta_{ij}\}_{j\in\mathcal{P}}$, *i.e.*, $\delta_{ii} := k_i\gamma_i + \theta_{ii}$ and $\delta_{ij} := \alpha_{ij} + \beta_{ji} + \theta_i j$ for $j \neq i$ as normal, where α_{ij} is obtained by decrypting the ciphertext $c_{\alpha_{ij}}$ and $\beta_{ji} := \gamma_i k_j - \alpha_{ji}$ (note that $\mathcal{B}_{ji} \neq g^{\beta_{ji}}$ in the simulation). After that, $\mathcal{F}$ computes D_i according to the verification equation, *i.e.*, $D_i := g^{\delta_{i,\mathcal{P}}} \prod_{j\in\mathcal{P}\setminus\{i\}} (\mathcal{B}_{ij}/\mathcal{B}_{ji})^{L_{j,\mathcal{P}}}$, so it must pass the verification.

3) Simulation of Online Signing Protocol: $\mathcal{F}$ aims to mold signature σ in $\mathcal{S}_{\mathsf{TSign}}$ as (r, s) received from $\mathcal{O}^{\mathsf{Sign}}$. $\mathcal{F}$ will simulate the threshold signatures $\{s_{ij}\}_{j\in\mathcal{P}^*}$ with respect to (r, s), and then the verification material $\{\mathcal{M}_{ij}\}_{j\in\mathcal{P}^*}$ according to the verification equation (Equation (3)) with respect to the simulated threshold signatures via $\mathcal{S}_{\mathsf{TSign}}$ as below.

Simulation of the threshold signature involves two steps. $\mathcal{F}$ first simulates shares with respect to $\{k_j\}_{j\in\mathcal{C}}$ and $\{x_j\}_{j\in\mathcal{C}}$. $\mathcal{F}$ then computes the additive shares $\mu_{ij} + \nu_{ji}$ and constructs the threshold signature using the simulated additive shares.

Since no one knows the actual values of k', the exponent of R and x' in signature $s = k'(m + rx')$ from $\mathcal{O}^{\mathsf{Sign}}$, $\mathcal{F}$ can pick $a \leftarrow\$ \mathbb{F}_q$ and set $b := (s/a - m)/r$. $\mathcal{F}$ then simulates randomness k'_i (simulated value with respect to the exponent of R) and key x'_i such that $a = \sum_{j\in\mathcal{C}} L_{j,\mathcal{P}^*} k_j + \sum_{j\in\mathcal{H}} L_{j,\mathcal{P}^*} k'_j$ and $b = \sum_{j\in\mathcal{C}} L_{j,\mathcal{P}^*} x_j + \sum_{j\in\mathcal{H}} L_{j,\mathcal{P}^*} x'_j$ as if $(k, x) = (a, b)$.

$\mathcal{F}$ computes $\mu_{ij} + \nu_{ji}$, which forms the threshold signature. According to the evaluation (equation of signature's reconstruction), correct reconstruction requires $(\mu_{ij} + \nu_{ji}) + (\mu_{ji} + \nu_{ij}) = k_i x_j + k_j x_i$ (we omit $'$ in $\{k'_i, x'_i\}_{i\in\mathcal{H}}$ for neater presentation). Note that μ_{ji} is sampled by party i and ν_{ij} is inversely computed (after decryption) during Phase 2, so $\mathcal{F}$ can compute $\mu_{ij} + \nu_{ji} := (k_i x_j + k_j x_i) - (\mu_{ji} + \nu_{ij})$.

$\mathcal{F}$ computes the threshold signature $\{s_{ij}\}_{j\in\mathcal{P}^*} := \{k_i m_{ii} + r k_i x_i, \{k_i m_{ij} + r(\mu_{ij} + \nu_{ji})\}_{j\in\mathcal{P}^*\setminus\{i\}}\}$ as normal, where $\{m_{ij}\}_{j\in\mathcal{P}^*}$ are Shamir secret sharing of m.

After the computation of the threshold signature, $\mathcal{F}$ simulates the verification material $\{\mathcal{M}_{ij}\}_{j\in\mathcal{C}}$ and $\{\mathcal{M}_{il}\}_{l\in\mathcal{H}}$ for the threshold signature sequentially. $\mathcal{F}$ computes:

- $\{\mathcal{M}_{ij}\}_{j\in\mathcal{C}}$: Evaluating verification equation with set $\mathcal{E} = \mathcal{C} \cup \{i\}$, which gives $X_j/(\prod_{\ell\in\mathcal{C}} R^{L_{\ell,\mathcal{E}} k_\ell x_j}) = R^{L_{i,\mathcal{E}} k_i x_j}$ to simulate $R^{k_i x_j}$, then compute $\mathcal{M}_{ij}$ via $\mathcal{M}_{ij} R^{\nu_{ij}} = R^{k_i x_j}$.
- $\{\mathcal{M}_{il}\}_{l\in\mathcal{H}}$: By $\{\mathcal{M}_{ij}\}_{j\in\mathcal{C}}$ obtained above, evaluating verification equation with set $\mathcal{E} = \mathcal{C} \cup \{i, l\}$ outputs two equations with two unknowns $\mathcal{M}_{li}$ and $\mathcal{M}_{il}$.
  - Evaluating the equation with respect to $R_i^m X_i^r$ gives an equation in term of $\mathcal{M}_{li}/\mathcal{M}_{il}$: $(R^{s_{il}} \mathcal{M}_{li}^r / \mathcal{M}_{il}^r)^{L_{l,\mathcal{E}}} = R_i^m X_i^r / (R^{L_{i,\mathcal{E}} s_{ii}} \prod_{j\in\mathcal{C}} (R^{s_{ij}} \mathcal{M}_{ji}^r / \mathcal{M}_{ij}^r)^{L_{j,\mathcal{E}}})$.
  - Evaluating the equation with respect to $R_l^m X_l^r$ gives an equation in terms of $\mathcal{M}_{li}/\mathcal{M}_{il}$: $(R^{s_{li}} \mathcal{M}_{il}^r / \mathcal{M}_{li}^r)^{L_{i,\mathcal{E}}} = R_l^m X_l^r / (R^{L_{l,\mathcal{E}} s_{ll}} \prod_{j\in\mathcal{C}} (R^{s_{lj}} \mathcal{M}_{jl}^r / \mathcal{M}_{lj}^r)^{L_{j,\mathcal{E}}})$.

  $\mathcal{F}$ solves for $\mathcal{M}_{i\ell}$ and $\mathcal{M}_{\ell i}$ simultaneously.

The unknowns are uniquely determined by the following values; the first two kinds are simulated, and the last is given:

- Threshold signature of honest party i $\{s_{i\ell}\}_{\ell\in\mathcal{P}^*}$;
- Verification material of party i for corrupted party j $\{\mathcal{M}_{ij}\}_{i\in\mathcal{H}, j\in\mathcal{C}}$;
- Verification material of corrupted party j: $\{\mathcal{M}_{j\ell}\}_{\ell\in\mathcal{P}^*}$.

Lemma 5: $\mathcal{S}_{\mathsf{TSign}}$ is indistinguishable from TSign.

Proof: The simulation differs from the real one in how threshold signature $\{s_{ij}\}_{j\in\mathcal{P}^*}$ and verification material $\{\mathcal{M}_{ij}\}_{j\in\mathcal{P}^*}$ are simulated. $\{s_{ij}\}_{j\in\mathcal{P}^*}$ is simulated using the received signature s from $\mathcal{O}^{\mathsf{Sign}}$. Our simulation writes $s := ab$ (for simplicity) and then simulates shares k_i, x_i with respect to a, b, which are identically distributed as the real ones. Thus, the (simplified) threshold signature $\{k_i x_i, (k_i x_j + k_j x_i) - (\mu_{ji} + \nu_{ij})\}$ has the same distribution as the real one. The verification material $\{\mathcal{M}_{ij}\}_{j\in\mathcal{P}^*}$ is simulated using the verification equation, so it naturally passes the verification. The indistinguishability of the simulated ZKP $\{\pi_{\mathcal{M}_{i\ell}}\}_{\ell\in\mathcal{P}^*}$ follows the zero-knowledge property of ZKP. ■

Appendix D
Open Problems

Our work leaves a few research problems that still need to be solved. Our DRG protocol and the MtAwc protocol require an additional communication round to issue complaint. Is there any more efficient approach? In particular, the possibility of a two-round distributed randomness generation protocol that features cheater identification and self-healing remains open.

ZKP is a vital ingredient in our threshold ECDSA scheme or generally secure multi-party computation. ZKP techniques could be made better in supporting witness extraction for the class groups. For example, Castagnos *et al.* [6], [7] rely on the low-order assumption and strong root assumption to extract the witness in the order-q group, but not the exponent in the hidden order group, usually storing the randomness or decryption key. Extracting witnesses in the hidden order group either requires a new assumption in the generic group model [27] or a binary challenge space [5]. The former asks for a closer inspection of the assumption in an idealized model, while the latter requires a parallel repetition of ZKP to reduce the soundness error. It is an open problem to devise an efficient ZKP protocol with special soundness for the hidden order group without additional assumptions in an idealized model.