Three-Round (Robust) Threshold ECDSA from Threshold CL Encryption

Bowen Jiang, Guofeng Tang$^{(\boxtimes)}$, and Haiyang Xue

School of Computing and Information Systems, Singapore Management University, Singapore 188065, Singapore
bowen.jiang.2024@phdcs.smu.edu.sg
{tang.guofeng789,haiyangxc}@gmail.com

Abstract. Threshold ECDSA has become a crucial security component in blockchain and decentralized systems, as it mitigates the risk of a single point of failure. Following the multiplicative-to-additive approach, the state-of-the-art threshold ECDSA (Doerner et al. in S&P24) requires only three rounds but has $O(n)$ outgoing communication complexity. Based on threshold CL encryption, Wong et al. (in NDSS24) proposed the first scheme with constant outgoing communication; however, their scheme requires at least four rounds.
We bridge this gap by introducing a three-round threshold ECDSA scheme with constant outgoing communication based on threshold CL encryption. Additionally, we enhance our basic scheme with robustness while maintaining the number of communication rounds, albeit at the cost of non-constant outgoing communication. Our implementation demonstrates that the basic scheme achieves optimal runtime and communication costs, while the robust variant reduces the communication rounds required by Wong et al.'s scheme, incurring only a small additional cost in small-scale settings.

Keywords: Threshold ECDSA · Three-Round Communication · Threshold CL encryption · Robustness.

1 Introduction

Threshold signatures [13, 14] allow a group of parties to generate signatures in a distributed way, where any subset of $t+1$ out of n parties can collaboratively produce a valid signature. Because of tolerating at most t dishonest parties, the threshold signature scheme has become an essential security component in blockchain and decentralized systems (Bitcoin [24] and Ethereum [30]), protecting against private key compromise by eliminating single points of failure. Common signature schemes used in these systems include ECDSA, Schnorr, and BLS. The importance of threshold signatures extends beyond the blockchain community. Recognizing the risks posed by single points of failure, the National Institute of Standards and Technology (NIST) launched a standardization initiative in 2020 for Multi-Party Threshold Cryptography [3]. This effort focuses on developing threshold variants of widely used signature schemes (such as ECDSA and EdDSA), with an emphasis on formal security definitions and practical efficiency.

Due to the wide deployment of ECDSA in decentralized systems, we focus on thresholding ECDSA in this paper. The design of threshold ECDSA presents significant cryptographic challenges due to its inherently non-linear signing equation. Unlike Schnorr and EdDSA signatures that allow straightforward linear threshold constructions, the signing algorithm of ECDSA requires more complex non-linear operations. Specifically, the signature equation

$$s = k^{-1}(H(m) + xr) \mod q \tag{1}$$

where k denotes the ephemeral nonce, x represents the private key, and r is the x-coordinate of elliptic curve point $R = g^k$. The equation shows that signing operations involve both the multiplicative inverse and the multiplication of secret values. In threshold ECDSA, these operations must be performed on secret-shared values. Thus, it requires secure distributed subprotocols to calculate both k^{-1} and $k^{-1}x$ from secret shares of k and x.

MtA Approaches. In recent years, numerous threshold ECDSA protocols have aimed to overcome this challenge primarily through Multiplication-to-Addition (MtA) operations [5, 8, 12, 15–17, 21, 23, 27, 31, 33, 34]. The core idea of MtA is to transform multiplication operations into addition operations through homomorphic properties of underlying cryptographic primitives and related zero-knowledge proofs.

Although several optimizations for MtA have been proposed in various settings—such as Oblivious Transfer [17], Paillier [27, 31], Castagnos-Laguillaumie (CL) encryption [8, 12, 29], and Joye-Libert [32]—MtA remains computationally and/or communicationally expensive, making it a dominant cost factor in threshold signing. Furthermore, the inherently two-party nature of MtA presents challenges in multi-party settings. Most existing approaches require pairwise MtA executions over point-to-point channels, leading to $O(n)$ communication complexity and $O(n^2)$ computational complexity.

The state-of-the-art in this area is the work by Doerner et al. [17], denoted by DKLs24, which introduces a three-round threshold signing protocol by reformulating the signature equation and leveraging VOLE for MPC operations. While the use of Oblivious Transfer-based MtA enables high computation efficiency, the scheme still incurs $O(n)$ communication complexity. Moreover, each OT-based MtA operation requires at least 50 KiB of communication [17, section 8.5], further contributing to the overall cost.

Threshold CL Approach. Castagnos-Laguillaumie (CL) encryption [9] is an additive homomorphic encryption scheme. One of its key advantages is its flexible message space, which can be freely adjusted to match that of ECDSA. Additionally, similar to ElGamal, it is well-suited for threshold decryption and related applications. Building on the work of Braun et al. [4], which explored multiparty computation using threshold CL encryption, Wong et al. [28] leveraged this technique to design a four-round threshold ECDSA protocol (hereafter denoted by WMC24). Their most significant contribution is the first threshold ECDSA scheme with constant communication complexity. Additionally, they achieved identifiable abort, which allows the detection of misbehaving parties, and robustness, ensuring that the signing process can continue and produce a valid signature as long as the remaining participants meet the threshold requirements.

Round Complexity and Communication Complexity. As in other multi-party computation protocols, communication rounds are a key efficiency metric in distributed signature protocols. If each round incurs a network latency of δ, the total delay scales to $O(r\delta)$, where r is the number of rounds. This effect is especially pronounced in geographically distributed deployments. Furthermore, reducing the number of rounds minimizes reliance on network stability, as each additional round introduces potential failure points where network interruptions could necessitate a protocol restart.

Motivation. In summary, while DKLs24 [17] operates in just three rounds, it requires at least $O(n)$ communication. On the other hand, WMC24 [28] achieves constant communication complexity and robustness but requires at least four rounds, leading to increased network delay, as previously discussed. These trade-offs motivate this work: Can we design a three-round threshold ECDSA scheme while maintaining constant communication complexity? Moreover, what must be sacrificed if we aim to achieve both three-round execution and robustness?

1.1 Our Contribution

We introduce two three-round threshold ECDSA schemes based on threshold CL encryption, comprising a two-round offline signing phase followed by a non-interactive online signing phase. The first scheme, TECDSA-Normal, serves as a basic non-robust variant, while the second, TECDSA-Robust, extends it to incorporate robustness.

The comparison of previous threshold ECDSA protocols and ours is given in Table 1 in terms of rounds, computation, and (outgoing) communication. Our basic scheme (without robustness) achieves three-round signing while maintaining constant communication complexity. Meanwhile, TECDSA-Robust achieves both three-round signing and robustness at the cost of sacrificing constant communication complexity.

We implemented both threshold ECDSA schemes in C++ using the BICYCL library. The experimental results are illustrated in Figure 5 and Table 2. Our basic scheme without robustness has the best performance. With $n = 20$ participants, our robust scheme reduces the execution time by 50% and offline communication overhead by 23% compared to WMY23 [29]. In small-scale settings ($n = 2, 3, 5$), it maintains comparable efficiency with WMC24 [28]. With one fewer offline round than WMC24, our robust protocol has the potential to achieve better efficiency in network environments where communication latency is substantial.

Signing Protocols	Rounds	Computation	Communication	Fault ID	Robust
CCL23 [8]	7	$O(n) + O(n^2)$	$O(n) + O(n)$	✓	×
WMY23 [29]	6	$O(n) + O(tn^2)$	$O(n) + O(n)$	✓	✓
WMC24 [28]	4	$O(n) + O(n)$	$\boldsymbol{O(1)}$	✓	✓
DKLs24 [17]	**3**	$O(n)$	$O(n)$	×	×
TECDSA-Normal	**3**	$O(n) + O(n)$	$\boldsymbol{O(1)}$	✓	×
TECDSA-Robust	**3**	$O(n) + O(tn)$	$O(t)$	✓	✓

Table 1: Round, computation and (outgoing) communication of (t, n)-threshold ECDSAs. "+" denotes the extra cost for fault identification (Fault ID). Fault ID is the foundation of robustness: identifying misbehaving participants and discarding their contributions is essential to achieving robustness.

1.2 Technical Overview

We begin by analyzing why WMC24 [28] requires four rounds. We then present our three-round approach without robustness, followed by an enhanced solution that incorporates robustness.

Let $\mathsf{ElG.Enc}(g^a)$ denote the ElGamal encryption of g^a. It satisfies the property $\mathsf{ElG.Enc}(g^a) \cdot \mathsf{ElG.Enc}(g^b) = \mathsf{ElG.Enc}(g^{a+b})$ for some publicly defined operation $\cdot$. CL encryption is an additively homomorphic encryption scheme with a structure similar to ElGamal encryption in the exponent—meaning the plaintext is a rather than g^a. Let $\mathsf{CL.Enc}(a)$ denote the CL encryption of a. It satisfies $\mathsf{CL.Enc}(a) \oplus \mathsf{CL.Enc}(b) = \mathsf{CL.Enc}(a+b)$ for some public operation $\oplus$, which differs from that in ElGamal encryption.

WMC24's Approach. WMC24 rewrote the form of ECDSA as $s = k(H(m) + xr)$ and $R = g^{k^{-1}}$. Their scheme works as follows; in the first three rounds prepare components for the final round of online signing.

- **Round 1:** Compute the CL encryption $\mathsf{CL.Enc}(k_i)$ of the shared value k_i and the ElGamal encryption $\mathsf{ElG.Enc}(g^{\phi_i})$ for the shared randomness ϕ_i. Using the homomorphic properties, derive $\mathsf{CL.Enc}(k)$ and $\mathsf{ElG.Enc}(g^{\phi})$.
- **Round 2:** Compute $\mathsf{CL.Enc}(k \cdot x_i)$ and $\mathsf{CL.Enc}(k \cdot \phi_i)$, which collectively yield $\mathsf{CL.Enc}(k \cdot x)$ and $\mathsf{CL.Enc}(k \cdot \phi)$.
- **Round 3:** Perform threshold decryption on $\mathsf{CL.Enc}(k \cdot \phi)$ to obtain $k \cdot \phi$. Similarly, threshold decrypt $\mathsf{ElG.Enc}(g^{\phi})$ to derive g^{ϕ}.
- **Round 4:** Construct and threshold decrypt $\mathsf{CL.Enc}(k(H(m)+rx))$ to extract $s = k(H(m) + rx)$.

$R = g^{k^{-1}}$ can be computed using $(g^{\phi})^{(k \cdot \phi)^{-1}}$ only after at least three rounds of communication. The value r is then derived from R and used to construct $\mathsf{CL.Enc}(k(H(m) + rx))$, enabling the extraction of $s = k(H(m) + rx)$ in the fourth round. Since $R = g^{k^{-1}}$ can only be obtained after three rounds, WMC24 needs at least four rounds, even if robustness and/or fault identification are not required.

Our Approach. We start from the original form as defined in Equation 1. Let $\mathsf{Com}(a)$ be a commitment of a. Our first scheme utilizes additive secret sharing and roughly works as follows:

– **Round 1:** Compute the CL encryption $\mathsf{CL.Enc}(\phi_i)$ of the shared randomness ϕ_i and the commitment $\mathsf{Com}(g^{k_i})$ for the shared randomness ϕ_i. Using the homomorphic property, derive $\mathsf{CL.Enc}(\phi)$.
– **Round 2:** Compute $\mathsf{CL.Enc}(\phi x_i)$ and $\mathsf{CL.Enc}(\phi k_i)$ for shared secret key x_i and shared randomness k_i, which collectively yield $\mathsf{CL.Enc}(\phi x)$ and $\mathsf{CL.Enc}(\phi k)$. Open the commitment $\mathsf{Com}(g^{k_i})$ to obtain $R = g^k$ and subsequently derive r.
– **Final Online Round:** Upon receiving message m, reconstruct the ciphertext $C = \mathsf{CL.Enc}(\phi(H(m) + rx))$ using the homomorphic property. Perform threshold decryption on C and $\mathsf{CL.Enc}(\phi k)$ to compute s as

$$s = \frac{\phi(H(m) + rx)}{\phi k} = k^{-1}(H(m) + rx).$$

In this way, we reduce the communication rounds to three while maintaining constant communication complexity. To defend against malicious adversaries, several zero-knowledge proofs are required. For details, please refer to Section 3.

Robustness. However, the above scheme is not robust. The reason is as follows. Each party picks the secret share k_i randomly, implicitly assembling the secret $k = k_1 + \cdots + k_n$. If some party P_{i^*} does not contribute correct k_{i^*} in Round 2, the protocol will get $R = g^{k - k_{i^*}}$. That is, the adversary can bias the distribution of R. The above argument can also be reflected in the security proof: even if the simulator can extract the adversary's R_js from $\mathsf{Com}(R_j)$ after Round 1, the simulator *cannot* simulate honest parties' R_is s.t. all R_is and R_js sum up to a fixed R obtained from the ECDSA oracle if some party P_{j^*} chooses not to correctly open R_{j^*} in Round 2. Non-robust schemes [5, 17] can tolerate this because the protocol will abort if a party fails to correctly open its commitment, but a robust protocol will continue.

In the second approach, we convert the n-out-of-n share k_i into t-out-of-n share, making k_i t-threshold[1]. In this case, the honest parties' R_i's are simulatable, which are t-threshold shares of the pre-determined R. A straightforward approach is to use Shamir's secret sharing. However, on this basis, we also need to ensure that the correctness of the distributed shares is publicly verifiable, where malicious parties can be publicly identified and excluded. At the same time, we aim to avoid introducing additional rounds for the generation of t-threshold k_i. To achieve these properties, we borrow [6]'s two-round distributed randomness generation (DRG) protocol with publicly verifiable secret sharing (PVSS) as a basic tool. The current two offline rounds and DRG can be executed concurrently without increasing the number of rounds. For details, please refer to Section 4.

[1] We say $\{k_i\}_{i\in[n]}$ are t-threshold, if any n' such shares can reconstruct k if $n' > t$.

1.3 Related Work

Based on MtA. At present, the construction of MtA can be implemented by various primitives, including Paillier homomorphic encryption [21,23], Castagnos-Laguillaumie (CL) encryption [7, 10, 12, 29, 34], Joye-Libert (JL) homomorphic encryption [32], and via oblivious transfer (OT) [15–17].

Doerner et al. [15] presented a three-round 2-out-of-n threshold ECDSA scheme that MtA is instantiated by an actively-secure variant of Gilboa's OT-based multiplication protocol [20]. In their subsequent work, [16] constructed a three-round variant with reduced bandwidth requirements. The scheme introduces a $\log t$-round protocol that converts t-party multiplicative shares to additive shares, enabling extension to the multi-party setting. Their threshold construction requires communication bandwidth of $50t$ KBytes per party. Building upon [16, 17] devised a novel two-round VOLE protocol to construct an OT-based t-out-of-n three-round threshold ECDSA scheme without zero-knowledge proofs. Their work achieves lower communication costs than [15, 16], and introduces no additional assumptions or primitives.

Based on Threshold CL Encryption. Braun et al. [4] constructed the first threshold scheme for CL encryption by designing zero-knowledge proofs for multiplicative relations between encrypted values. Their construction enables constant-communication secure multiparty computation in the You-Only-Speak-Once (YOSO) model. Building on this work, Wong et al. [28] applied threshold CL encryption to threshold ECDSA, achieving protocols with constant communication complexity, while existing protocols based on MtA [8, 17] require quadratic communication.

Others. Abram et al. [1] developed a threshold ECDSA with very low bandwidth using a pseudo-random correlation generator (PCG). However, the computational cost of their scheme is high (i.e., 1 to 2 seconds per ECDSA signature).

2 Preliminaries

Let $[n]$ denote the set $\{1, 2, \cdots, n\}$, where n is the number of parties indexed by $i \in [n]$. The system has a corruption threshold t and a computational security parameter λ, where all parties run in polynomial time with respect to λ. Let $\oplus$ and $\odot$ denote the homomorphic addition and scalar multiplication operations on ciphertexts, respectively. For any subset $\mathcal{S}$ of parties, the Lagrange coefficient $\ell_i^{\mathcal{S}}$ for party $i \in \mathcal{S}$ evaluated at $x = 0$ is computed as $\ell_i^{\mathcal{S}} = \prod_{j \in \mathcal{S} \setminus i} \frac{j}{j-i}$. Let $\overline{a}$ denote the CL ciphertext of $a \in \mathbb{Z}_q$.

2.1 The ECDSA Signature

Let $\mathbb{G} =< g >$ be an elliptic curve group of order q with generator g. Denote the public parameters as $\mathsf{pp}_{\mathsf{Sig}} = (\mathbb{G}, g, q)$. A cryptography hash function is defined as $H : \{0, 1\}^* \to \mathbb{Z}_q$. The ECDSA is defined as follows:

1. $\mathsf{Keygen}(1^\lambda)$: on input 1^λ, choose a random $x \leftarrow \mathbb{Z}_q$, set x as the private key. Compute $X = g^x$, and set X as the public key.
2. $\mathsf{Sign}(x, m)$: on input sign key x and message m
   - Choose a random $k \leftarrow \mathbb{Z}_q$, compute $R = (r_x, r_y) = g^k$.
   - Compute $r = r_x \mod q$ and $s = k^{-1}(H(m) + rx) \mod q$.
   - Output (r, s) as the signature [2].
3. $\mathsf{Verify}(m; (r, s))$ calculates $(r_x, r_y) = R = g^{s^{-1}H(m)} \cdot X^{s^{-1}r}$ and outputs 1 if and only if $r = r_x \mod q$.

The ideal functionality $\mathcal{F}_{\mathsf{ECDSA}}$ for threshold ECDSA is shown in Figure 1. It consists of two functions, namely, a key generation function Keygen, called once, and a signing function Sign, called an arbitrary number of times under the generated key. We, following [23], say that an (t, n)-threshold ECDSA is secure if it realizes $\mathcal{F}_{\mathsf{ECDSA}}$ in the presence of a malicious static adversary according to the real/ideal definition [22].

Functionality $\mathcal{F}_{\mathsf{ECDSA}}$ interacts with parties $P_1, \ldots, P_n$ as follows:

- Key Generation Keygen: On receiving $\mathsf{Keygen}(1^\lambda)$ from all parties,
  - select a random value $x \leftarrow \mathbb{Z}_q$ and computes the public key $X = g^x$
  - sent X to $P_1, \ldots, P_n$
  - store $(\mathbb{G}, g, q, x, X)$ and ignore any further call to Keygen
- Signature Generation Sign: Upon receiving $(\mathsf{sign}, \mathsf{sid}, m)$ from at least $t + 1$ distinct parties, if sid has not been used before,
  - select a random value $k \leftarrow \mathbb{Z}_q$ and compute $R = (r_x, r_y) = g^k$.
  - calculate $r = r_x \bmod q$ and $s = k^{-1}(H(m) + rx) \bmod q$
  - send (r, s) to the queried parties and store $(\mathsf{complete}, \mathsf{sid})$

Fig. 1: The (t, n)-threshold ECDSA functionality

2.2 CL Threshold Encryption

The class group framework [9] is defined by public parameters $(\widetilde{s}, f, G^q, g_q, \widehat{G}, F, q)$, where $\widehat{G}$ is a finite abelian group of order $q\hat{s}$. The factor $\hat{s}$ remains unknown with $(q, \hat{s}) = 1$ and $\widetilde{s}$ serves as an upper bound for $\hat{s}$, ensuring $s|\hat{s}$. Three subgroups are defined based on $\widehat{G}$:

- $F = \langle f \rangle$, a unique subgroup of order q.
- $G^q = \langle g_q \rangle$, a subgroup of q-th powers with order s.
- $G' = \langle f g_q \rangle$, a cyclic subgroup of order qs.

[2] It is well known that for every valid signature (r, s), the pair $(r, -s)$ is also a valid signature. To make (r, s) unique, in this paper, we mandate that the "smaller" of $\{s, -s\}$ is the output.

These subgroups are related by the isomorphism $F \times G^q \simeq G' \subset \widehat{G}$.

The Castagnos-Laguillaumie encryption. Let $\mathsf{pp}_{\mathsf{CL}} := (\widetilde{s}, f, G^q, g_q, \widehat{G}, F, q)$ be the public parameters, and let $\mathcal{D}_q$ denote the interval $[0, 2^{\lambda} q \widetilde{s}]$. The Castagnos-Laguillaumie (CL) encryption scheme [9] operates as follows.

- $\mathsf{CL.KGen}(\mathsf{pp}_{\mathsf{CL}}, 1^{\lambda}) \rightarrow (\mathsf{pk}, \mathsf{sk})$: Randomly choose $\mathsf{sk} \leftarrow\!\$\; \mathcal{D}_q$ and $\mathsf{pk} := g_q^{\mathsf{sk}}$.
- $\mathsf{CL.Enc}(\mathsf{pk}, m) \rightarrow (c_1, c_2)$: Given pk and m, randomly choose $r \leftarrow\!\$\; \mathcal{D}_q$, output ciphertext $(c_1, c_2) = (g_q^r, f^m \mathsf{pk}^r)$.
- $\mathsf{CL.Dec}(\mathsf{sk}, (c_1, c_2)) \rightarrow m$: Given sk and c, output $m = \log_f(c_2 / c_1^{\mathsf{sk}})$.

The threshold CL encryption. A t-out-of-n threshold CL scheme [4, 28], requiring $t+1$ quorum for decryption, comprises five probabilistic polynomial time (PPT) algorithms, denoted as $\mathsf{t\text{-}CL} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{PartDec}, \mathsf{FinDec})$. The Setup algorithm generates the public parameters tuple $\mathsf{pp}_{\mathsf{CL}} = (\widetilde{s}, f, G^q, g_q, \widehat{G}, F, q)$ for the CL framework [9, 10].

Upon completion of key generation KGen in [4, 28], each party P_i stores a private key share $\mathsf{sk}_i \in \mathcal{D}_q$ and the corresponding public key share $\mathsf{pk}_i = g_q^{\Delta \mathsf{sk}_i}$, while the aggregate public key is computed as $\mathsf{pk} = g_q^{\Delta \mathsf{sk}}$, where $\mathsf{sk} = \sum_{i \in \mathcal{S}} \ell_i^{\mathcal{S}} \mathsf{sk}_i$. The encryption algorithm Enc follows the original CL encryption algorithm with a minor modification: it replaces g_q with $g_q^{\Delta^2}$ in the first part of the ciphertext. For decryption, the process is split into two phases, $\mathsf{PartDec}$ and FinDec illustrated in the following.

- $\mathsf{t\text{-}CL.Enc}(\mathsf{pk}, m) \rightarrow c$: Given pk and message m, output $c = (c_1, c_2) = ((g_q^{\Delta^2})^r, f^m \mathsf{pk}^r)$.
- $\mathsf{t\text{-}CL.PartDec}(\mathsf{pk}, \mathsf{sk}_i, c) \rightarrow (pc_i, \pi_i)$: Given pk, sk_i, and $c = (c_1, c_2)$, output partial decryption $pc_i = c_1^{\Delta \mathsf{sk}_i}$ and a zero-knowledge proof of partial decryption π_i (which will given in section 2.3).
- $\mathsf{t\text{-}CL.FinDec}(\mathsf{pk}, \{pc_i, \pi_i\}_{i \in \mathcal{S}}, c) \rightarrow m$: Given pk, a set $\{pc_i, \pi_i\}_{i \in \mathcal{S}}$ with $|\mathcal{S}| \geq t+1$, and $c = (c_1, c_2)$, verify π_i for all $i \in \mathcal{S}$. If all the proofs are accepted, then compute and output $m = \frac{\log_f \left(c_2^{\Delta^2} / \prod_{i \in \mathcal{S}} pc_i \right)}{\Delta^2} \bmod q$, otherwise abort.

We need the following security properties of CL and $\mathsf{t\text{-}CL}$ for our schemes.

- CL encryption is indistinguishable under chosen plaintext attacks (IND-CPA) secure under the DDH assumption over $\widehat{G}$ [9, Theorem 4]
- We say $\mathsf{t\text{-}CL}$ is t-IND-CPA secure if it is IND-CPA secure even allowing the adversary to corrupt at most t parties [28].
- $\mathsf{t\text{-}CL}$ satisfies simulation security which says, without knowing the secret key sk_i, we still can simulate pc_i and π_i which is indistinguishable from real ones [28, Lemma 7].

2.3 Zero-Knowledge Proof

An interactive proof for a language L is an interactive protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. Assume $\mathcal{R}$ is the associated relation of L. We call $(\mathcal{P}, \mathcal{V})$

an interactive proof for $\mathcal{R}$ or L if it satisfies: completeness which says for every $x \in L$, $\langle \mathcal{P}, \mathcal{V} \rangle(x)$ always accepts; and soundness which says for every $x \notin L$ and every prover $\mathcal{P}^*$, $\Pr[\langle \mathcal{P}^*, \mathcal{V} \rangle(x) = 1]$ is negligible.

An interactive proof is zero-knowledge if for every PPT V^* there exists a PPT simulator Sim s. t. $\{View^{\mathcal{P}}_{\mathcal{V}^*}(x)\}_{x \in L}$ and $\{\mathsf{Sim}_{\mathcal{R}}(x)\}_{x \in L}$ are statistically indistinguishable. It is said to be honest-verifier zero-knowledge if zero-knowledge holds for any PPT honest verifier.

Σ-protocol [11] is a special honest-verifier zero-knowledge proof. Σ-protocol can be transformed into a non-interactive zero-knowledge proof (NIZK) using the Fiat-Shamir transform [18] in the random oracle model. To simplify the presentation, we denote NIZK for relation $\mathcal{R}$ as $\Pi^{\mathcal{R}}_{\mathsf{zk}} = (\mathsf{Prove}_{\mathcal{R}}, \mathsf{Vrfy}_{\mathcal{R}})$:

- $\mathsf{Prove}_{\mathcal{R}}(x, w) \to \pi_{\mathcal{R}}$: Given a statement x and a witness w, output $\pi_{\mathcal{R}}$.
- $\mathsf{Vrfy}_{\mathcal{R}}(x, \pi_{\mathcal{R}}) \to b$: Given a statement x and a proof $\pi_{\mathcal{R}}$, output $b \in \{0, 1\}$ indicating $\pi_{\mathcal{R}}$ is accepted or rejected.

A zero-knowledge proof of knowledge is a stronger form of zero-knowledge proof that not only demonstrates that $x \in L$ but also requires a proof of knowledge. This means there must exist a PPT knowledge extractor Ext that can extract the witness w from a valid proof. A better way to capture this is ideal zero-knowledge functionality. The ideal zero-knowledge functionality $\mathcal{F}^{\mathcal{R}}_{\mathsf{zk}}$ within binary relation $\mathcal{R}$ is defined in Figure 2.

Functionality $\mathcal{F}^{\mathcal{R}}_{\mathsf{zk}}$ interacts with parties $P_1, \cdots P_n$ as follows:
Upon receiving $(\mathsf{prove}, \mathsf{sid}, i, x, w)$ from party P_i where $i \in [n]$, $\mathcal{F}^{\mathcal{R}}_{\mathsf{zk}}$ checks if sid has been previously used. If not, $\mathcal{F}^{\mathcal{R}}_{\mathsf{zk}}$ broadcasts $(\mathsf{proof}, \mathsf{sid}, x)$ to all parties if $\mathcal{R}(x, w) = 1$; otherwise, ignores the message.

Fig. 2: The zero-knowledge functionality

The committed NIZK functionality aims to capture the commitment of non-interactive zero-knowledge proof of knowledge. It, denoted by $\mathcal{F}^{\mathcal{R}}_{\mathsf{com\text{-}zk}}$, is illustrated in Figure 3.

We will need zero-knowledge proofs, ideal zero-knowledge proof functionality, or ideal committed NIZK for the following relations. To be specific, we need zero-knowledge proofs for relations $\mathcal{R}_{\mathsf{dl}}$, $\mathcal{R}_{\mathsf{dl\text{-}cl}}$, $\mathcal{R}_{\mathsf{part\text{-}dec}}$, $\mathcal{R}_{\mathsf{sh}}$, and $\mathcal{R}_{\mathsf{cl\text{-}dec\text{-}dl}}$, $\mathcal{F}^{\mathcal{R}}_{\mathsf{zk}}$ for $\mathcal{R}_{\mathsf{cl\text{-}enc}}$, and $\mathcal{F}^{\mathcal{R}}_{\mathsf{com\text{-}zk}}$ for $\mathcal{R}_{\mathsf{dl}}$.

Relation for Discrete Logarithm. $\mathcal{R}_{\mathsf{dl}}$ verifies that the witness $k \in \mathbb{Z}_q$ is the discrete logarithm of $R \in \mathbb{G}$. Refer to [26] for the instantiation.

$$\mathcal{R}_{\mathsf{dl}} = \left\{(R; k) \;\middle|\; R \in \mathbb{G}, k \in \mathbb{Z}_q, R = g^k\right\}$$

Relation for CL Encryption. $\mathcal{R}_{\mathsf{cl\text{-}enc}}$ verifies that a CL ciphertext $c = (c_1, c_2)$ is a correct format with message m with the random nonce r.

$$\mathcal{R}_{\mathsf{cl\text{-}enc}} = \left\{(\mathsf{pk}, c; m, r) \;\middle|\; r \in \mathcal{D}_q,\, c_1 = \left(g_q^{\Delta^2}\right)^r,\, c_2 = f^m \mathsf{pk}^r\right\}$$

Functionality $\mathcal{F}_{\textsf{com-zk}}^{\mathcal{R}}$ interacts with parties $P_1, \cdots, P_n$ as follows:

- Commit and Prove: Upon receiving $(\textsf{com-prove}, \textsf{sid}, x, w)$ from a party P_i where $i \in [n]$, $\mathcal{F}_{\textsf{com-zk}}^{\mathcal{R}}$ checks if $\textsf{sid}$ has been previously used. If not, $\mathcal{F}_{\textsf{com-zk}}^{\mathcal{R}}$ stores $(\textsf{sid}, i, x, w)$ and broadcasts $(\textsf{proof-receipt}, \textsf{sid}, i)$ to all parties; otherwise, $\mathcal{F}_{\textsf{com-zk}}^{\mathcal{R}}$ ignores the message.
- Decommit and Verify: Upon receiving $(\textsf{decom-proof}, \textsf{sid}, i)$ from P_i where $i \in [n]$, $\mathcal{F}_{\textsf{com-zk}}^{\mathcal{R}}$ first checks that a record $(\textsf{sid}, i, x, w)$ exists and $(\textsf{decom-proof}, \textsf{sid}, i)$ has not been received before. If these conditions are met, $\mathcal{F}_{\textsf{com-zk}}^{\mathcal{R}}$ checks whether $(x, w) \in \mathcal{R}$. If $(x, w) \in \mathcal{R}$, $\mathcal{F}_{\textsf{com-zk}}^{\mathcal{R}}$ broadcasts $(\textsf{decom-proof}, \textsf{sid}, x, 1)$ to all parties; otherwise, broadcasts $(\textsf{decom-proof}, \textsf{sid}, x, 0)$ to all parties.

Fig. 3: The Committed NIZK Functionality $\mathcal{F}_{\textsf{com-zk}}^{\mathcal{R}}$ for $\mathcal{R}$

Relation for Multiplying CL Ciphertext with Discrete Logarithm. $\mathcal{R}_{\textsf{dl-cl}}$ verifies that the CL ciphertext $c_1 = (c_{11}, c_{12})$ is obtained by multiplying $c_0 = (c_{01}, c_{02})$ with $x \in \mathbb{Z}_q$, where $X = g^x \in \mathbb{G}$.

$$\mathcal{R}_{\textsf{dl-cl}} = \{(X, c_0, c_1; x) \mid X = g^x, c_{01} = {c_{11}}^x, c_{02} = {c_{12}}^x\}$$

Relation for Correct Partial Decryption. $\mathcal{R}_{\textsf{part-dec}}$ verifies that pc_i is a partial decryption of $c = (c_1, c_2)$ that $pc_i = (c_1^{\Delta})^{\textsf{sk}_i}$, where $\textsf{sk}_i$ is a CL encryption key share.

$$\mathcal{R}_{\textsf{part-dec}} = \left\{(\textsf{pk}_i, c, pc_i; \textsf{sk}_i) \;\middle|\; \textsf{pk}_i = (g_q^{\Delta})^{\textsf{sk}_i}, pc_i = (c_1^{\Delta})^{\textsf{sk}_i}\right\}$$

For the instantiation of $\mathcal{R}_{\textsf{cl-enc}}$, $\mathcal{R}_{\textsf{dl-cl}}$, $\mathcal{R}_{\textsf{part-dec}}$, refer to [28, Appendix A].
Relation for Validity of Encrypted Shares. $\mathcal{R}_{\textsf{sh}}$ verifies that CL ciphertexts $\left\{C_{f(i)}\right\}_{i \in \mathcal{T}}$ are encryptions of a random polynomial $f(u) \in \mathbb{Z}_q[u]_{\le t}$ evaluations.

$$\mathcal{R}_{\textsf{sh}} = \left\{ (\{C_{f(i)}\}_{i \in \mathcal{T}}, f(u)) \;\middle|\; \begin{array}{l} f(u) \in \mathbb{Z}_q[u]_{\le t}, \\ \forall i \in \mathcal{T} : C_{f(i)} = \textsf{CL.Enc}(ek_i, f(i)) \end{array} \right\}$$

Relation for CL Decrypted Plaintext with Discrete Logarithm. $\mathcal{R}_{\textsf{cl-dec-dl}}$ verifies that a CL ciphertext C_a is encrypted by ek is a correct format with $a \in \mathbb{Z}_q$ and a is the discrete logarithm of A.

$$\mathcal{R}_{\textsf{cl-dec-dl}} = \{(C_a, A; a, dk) \mid a \in \mathbb{Z}_q,\ dk \in \mathcal{D}_q,\ A = g^a,\ a = \textsf{CL.Dec}(dk, C_a)\}$$

For the instantiation of $\mathcal{R}_{\textsf{sh}}$, $\mathcal{R}_{\textsf{cl-dec-dl}}$, refer to [6, Figure 4] and [4, Section 5].

3 Basic Threshold ECDSA: TECDSA-Normal

In this section, we present a three-round threshold ECDSA protocol $\Pi_{\textsf{TECDSA}}$ achieving t-out-of-n threshold security without the requirement of robustness.

3.1 Key Generation

The key generation incorporates two distinct distributed key generation (DKG) protocols: one for generating a key pair for CL threshold encryption, and another for ECDSA public key. Note that two DKG protocols are executed once in each call between n parties before the signing protocol is executed.

DKG for CL $\Pi_{\mathsf{DKG\text{-}CL}}$. The protocol is instantiated as in [4,28]. The protocol results in t-out-of-n access structure of key pair $(\mathsf{pk}; \{\mathsf{pk}_i\}_{i=1}^n, \{\mathsf{sk}_i\}_{i=1}^n)$ for the threshold CL encryption, where $(\mathsf{pk}_i = g_q^{\Delta \mathsf{sk}_i}, \mathsf{sk}_i)$ is the share of public-secret key pair for party P_i, and pk is the aggregate public key. Since it is the same as that [28], we leave the details of $\Pi_{\mathsf{DKG\text{-}CL}}$ in our full version.

DKG for ECDSA $\Pi_{\mathsf{DKG\text{-}Sig}}$. Since the distributed key generation of ECDSA key pair $(x, X = g^x)$ is a well-established problem, we can leverage existing works directly. It results in t-out-of-n access structure of key pair $(X; \{X_i\}_{i=1}^n, \{x_i\}_{i=1}^n)$, where $(X_i = g^{x_i}, x_i)$ is the share of public-secret key pair for party P_i, and X is the aggregated public key. Since it is the same as that in prior work [19], we leave the details in the full version.

3.2 Threshold Signing

After key generation, each party P_i obtains their private key sk_i and x_i, and corresponding public values $\{\mathsf{pk}_i\}_{j\in[n]}$ and $\{X_i\}_{i\in[n]}$ through the execution of $\Pi_{\mathsf{DKG\text{-}CL}}$ and $\Pi_{\mathsf{DKG\text{-}Sig}}$.

Π_{TECDSA} takes as input the system parameters $\left(\mathsf{pp}_{\mathsf{CL}}, \mathsf{pp}_{\mathsf{Sig}}, n, t, \mathcal{T}\right)$, where $\mathcal{T} \subset [n]$ and $|\mathcal{T}| \geq t+1$. For each $i \in \mathcal{T}$, P_i has the threshold CL encryption public key pk, the ECDSA public key X, a unique session identifier sid, the message m to be signed, a public key infrastructure comprising encryption keys $\{\mathsf{pk}_j\}_{j\in\mathcal{T}}$ and signing keys $\{X_j\}_{j\in\mathcal{T}}$, along with their respective secret shares consisting of a private signing key x_i and a decryption key sk_i.

Π_{TECDSA}, as illustrated in Fig. 4, comprises two offline rounds that can be executed independently of the message m, followed by an online signing round that incorporates m.

Round 1. Each party P_i acts as follows:

1. Sample $\phi_i, k_i \leftarrow\!\!\$\ \mathbb{Z}_q$ uniformly.
2. Calculate the ciphertext of threshold CL encryption $\bar{\phi}_i \leftarrow \mathsf{t\text{-}CL.Enc}(\mathsf{pk}, \phi_i)$.
3. Calculate the nonce share $R_i = g^{k_i}$.
4. Send $(\mathsf{prove}, \mathsf{sid}, i, \bar{\phi}_i, \phi_i)$ to $\mathcal{F}_{\mathsf{zk}}^{\mathcal{R}_{\mathsf{cl\text{-}enc}}}$ and $(\mathsf{com\text{-}prove}, \mathsf{sid}, i, R_i, k_i)$ to $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{dl}}}$.

Round 2. Upon receiving $(\mathsf{proof}, \mathsf{sid}, \bar{\phi}_j)$ from $\mathcal{F}_{\mathsf{zk}}^{\mathcal{R}_{\mathsf{cl\text{-}enc}}}$ and $(\mathsf{proof\text{-}receipt}, \mathsf{sid}, j)$ from $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{dl}}}$ for P_j, each party P_i acts as follows:

1. Use homomorphism to calculate $\bar{\phi} = \bigoplus_{j\in\mathcal{T}} \bar{\phi}_j$.

Three-Round Threshold ECDSA Protocol Π_{TECDSA}

P_i, where $i \in [n]$ **All** $\{P_j\}_{j \neq i, j \in [n]}$

Round 1 (sid)

Sample $\phi_i \leftarrow\!\!\$\ \mathbb{Z}_q, \bar{\phi}_i \leftarrow \mathsf{t\text{-}CL.Enc}(\mathsf{pk}, \phi_i)$ $\xrightarrow{(\mathsf{prove}, \mathsf{sid}, \bar{\phi}_i, \phi_i)} \mathcal{F}_{\mathsf{zk}}^{\mathcal{R}_{\mathsf{cl\text{-}enc}}} \xrightarrow{(\mathsf{prove}, \mathsf{sid}, \bar{\phi}_j)}$

Sample $k_i \leftarrow\!\!\$\ \mathbb{Z}_q, R_i = g^{k_i}$ $\xrightarrow{(\mathsf{com\text{-}prove}, \mathsf{sid}, i, R_i, k_i)} \mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{dl}}} \xrightarrow{(\mathsf{proof\text{-}receipt}, \mathsf{sid}, j)}$

Round 2 (sid)

$$\bar{\phi} = \bigoplus_{j \in \mathcal{T}} \bar{\phi}_j$$

Compute $\overline{\phi x_i} = \bar{\phi} \odot \ell_i^{\mathcal{T}} x_i$

$\pi^0_{\mathsf{dl\text{-}cl}\,i} \leftarrow \mathsf{Prove}_{\mathsf{dl\text{-}cl}}(X_i, \bar{\phi}, \overline{\phi x_i}; x_i)$ $\xrightarrow{\overline{\phi x_i}, \pi^0_{\mathsf{dl\text{-}cl}\,i}}$ $\mathsf{Vrfy}_{\mathsf{dl\text{-}cl}}(\mathsf{pp}_{\mathsf{CL}}, X_j, \bar{\phi}, \overline{\phi x_j}, \pi^0_{\mathsf{dl\text{-}cl}\,j})$

Compute $\overline{\phi k_i} = \bar{\phi} \odot k_i$

$\pi^1_{\mathsf{dl\text{-}cl}\,i} \leftarrow \mathsf{Prove}_{\mathsf{dl\text{-}cl}}(R_i, \bar{\phi}, \overline{\phi k_i}; k_i)$ $\xrightarrow{\overline{\phi k_i}, \pi^1_{\mathsf{dl\text{-}cl}_i}}$ $\mathsf{Vrfy}_{\mathsf{dl\text{-}cl}}(\mathsf{pp}_{\mathsf{CL}}, R_j, \bar{\phi}, \overline{\phi k_j}, \pi^1_{\mathsf{dl\text{-}cl}\,j})$

$\xrightarrow{(\mathsf{decom\text{-}proof}, \mathsf{sid}, i)} \mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{dl}}} \xrightarrow{(\mathsf{decom\text{-}proof}, \mathsf{sid}, R_j, b)}$ If $b = 0$, return $\perp$

Round 3 (sid, m)

$$R = \prod_{j \in \mathcal{T}} R_j, \text{Store } x\text{-coordinate of } R \text{ as } r_x$$

$$c^0 = \bigoplus_{j \in \mathcal{T}} \overline{\phi k_j}, c^1 = \bar{\phi} \odot H(m) \left\{ \bigoplus_{j \in \mathcal{T}} \overline{\phi x_j} \odot r_x \right\}$$

$(pc_i^0, \pi^0_{\mathsf{part\text{-}dec}\,i}) \leftarrow \mathsf{t\text{-}CL.PartDec}(\mathsf{pk}, \mathsf{sk}_i, c^0)$ $\xrightarrow{pc_i^0, \pi^0_{\mathsf{part\text{-}dec}\,i}}$ $\mathsf{Vrfy}_{\mathsf{part\text{-}dec}}(\mathsf{pk}, pc_i^0, c^0, \pi^0_{\mathsf{part\text{-}dec}\,j})$

$(pc_i^1, \pi^1_{\mathsf{part\text{-}dec}\,i}) \leftarrow \mathsf{t\text{-}CL.PartDec}(\mathsf{pk}, \mathsf{sk}_i, c^1)$ $\xrightarrow{pc_i^1, \pi^1_{\mathsf{part\text{-}dec}\,i}}$ $\mathsf{Vrfy}_{\mathsf{part\text{-}dec}}(\mathsf{pk}, pc_i^1, c^1, \pi^1_{\mathsf{part\text{-}dec}\,j})$

Output of Protocol

$p^0 \leftarrow \mathsf{t\text{-}CL.FinDec}(\mathsf{pk}, \{pc_i^0\}_{i \in \mathcal{T}}, c^0)$

$p^1 \leftarrow \mathsf{t\text{-}CL.FinDec}(\mathsf{pk}, \{pc_i^1\}_{i \in \mathcal{T}}, c^1)$

Output $(r, s) = (r_x, p^1/p^0)$

Fig. 4: Threshold ECDSA Π_{TECDSA}

2. Compute the ciphertext $\overline{\phi x_i} = \bar{\phi} \odot \ell_i^{\mathcal{T}} x_i$ and generate a zero-knowledge proof $\pi^0_{\mathsf{dl\text{-}cl}\,i} \leftarrow \mathsf{Prove}_{\mathsf{dl\text{-}cl}}(X_i, \bar{\phi}, \overline{\phi x_i}; x_i)$.

3. Compute the ciphertext $\overline{\phi k_i} = \bar{\phi} \odot k_i$ and generate a zero-knowledge proof $\pi^1_{\mathsf{dl\text{-}cl}i} \leftarrow \mathsf{Prove}_{\mathsf{dl\text{-}cl}}(R_i, \bar{\phi}, \overline{\phi k_i}; k_i)$.
4. Send $(\mathsf{decom\text{-}proof}, \mathsf{sid}, i)$ to $\mathcal{F}^{\mathcal{R}_{\mathsf{dl}}}_{\mathsf{com\text{-}zk}}$ and receive $(\mathsf{decom\text{-}proof}, \mathsf{sid}, R_j, b_j)$.
5. Broadcast $(\overline{\phi x_i}, \pi^0_{\mathsf{dl\text{-}cl}i})$ and $(\overline{\phi k_i}, \pi^1_{\mathsf{dl\text{-}cl}i})$ to all other parties.
6. Verify $\mathsf{Vrfy}_{\mathsf{dl\text{-}cl}}(\mathsf{pp}_{\mathsf{CL}}, X_j, \bar{\phi}, \overline{\phi x_j}, \pi^0_{\mathsf{dl\text{-}cl}j})$ and $\mathsf{Vrfy}_{\mathsf{dl\text{-}cl}}(\mathsf{pp}_{\mathsf{CL}}, R_j, \bar{\phi}, \overline{\phi k_j}, \pi^1_{\mathsf{dl\text{-}cl}j})$ for each $j \neq i$. If any verification fails or $b_j = 0$, P_i aborts.

Round 3. With the input of a message m, upon receiving $(\overline{\phi x_j}, \pi^0_{\mathsf{dl\text{-}cl}j})$ and $(\overline{\phi k_j}, \pi^1_{\mathsf{dl\text{-}cl}j})$ from P_j, each party P_i acts as follows:

1. Compute $R = \prod_{j \in \mathcal{T}} R_j$ and sets $r = r_x$ as the x-coordinate of R.
2. Locally compute $c^0 = \bigoplus_{j \in \mathcal{T}} \overline{\phi k_j}$.
3. Locally compute $\overline{\phi x r_x} = \bigoplus_{j \in \mathcal{T}} \overline{\phi x_j} \odot r_x$ and ciphertext $c^1 = \bar{\phi} \odot h \oplus \overline{\phi x r_x}$, where $h = H(m)$ is the output of hash function with inputting message m.
4. Broadcast the partially decrypted shares with its proof $(pc^0_i, \pi^0_{\mathsf{part\text{-}dec}_i}) \leftarrow \mathsf{t\text{-}CL.PartDec}(\mathsf{pk}, \mathsf{sk}_i, c^0)$ and $(pc^1_i, \pi^1_{\mathsf{part\text{-}dec}_i}) \leftarrow \mathsf{t\text{-}CL.PartDec}(\mathsf{pk}, \mathsf{sk}_i, c^1)$ to all other parties.

Upon receiving a threshold number of partially decrypted shares from other parties, P_i generates the signature (r, s) as:

1. Compute $p^0 \leftarrow \mathsf{t\text{-}CL.FinDec}(\mathsf{pk}, \{pc^0_i, \pi^0_{\mathsf{part\text{-}dec}_i}\}_{i \in \mathcal{T}}, c^0)$
2. Compute $p^1 \leftarrow \mathsf{t\text{-}CL.FinDec}(\mathsf{pk}, \{pc^1_i, \pi^1_{\mathsf{part\text{-}dec}_i}\}_{i \in \mathcal{T}}, c^1)$.
3. $s = p^1/p^0 \mod q$, output (r, s) if $\mathsf{Verify}(m; (r, s)) = 1$.

3.3 Security Proof of TECDSA-Normal

Theorem 1. *Assume that the parties $P_1, P_2, \dots, P_n$ are invoked with the same $(pp_{CL}, pp_{Sig}, n, t, \mathcal{T})$, where $\mathcal{T} \subset [n]$ contains at most t corrupt parties. If t-CL is simulation secure and t-IND-CPA secure, our threshold signature protocol securely realizes functionality $\mathcal{F}_{ECDSA}$ (defined in Fig. 1) in the presence of a malicious static adversary $\mathcal{A}$ in the $(\mathcal{F}^{\mathcal{R}}_{com\text{-}zk}, \mathcal{F}^{\mathcal{R}}_{zk})$-hybrid model.*

The simulator $\mathcal{S}$ interacts with the ideal functionality $\mathcal{F}_{\mathsf{ECDSA}}$ to generate ECDSA public key X, and many signature pairs (r, s). In the ideal world, $\mathcal{S}$ only learn X, and many $((r, s), m)$'s where m is the message to be signed. In the real-world adversary $\mathcal{A}$ observes all protocol interactions with honest parties. Therefore, given only the expected outputs X and $(r, s), m)$'s, $\mathcal{S}$ needs to simulate the view of adversary $\mathcal{A}$.

Let $\mathcal{A}$ be an adversary corrupting a subset of parties $I \subset [n]$ of size at most t, and let $J = \mathcal{T} \backslash I$ denote the set of honest parties. Without loss of generality, assume there exists one honest party $P_{j^*} \in J$.

Key Generation Phase. To simplify the proof, we take $\Pi_{\mathsf{DKG\text{-}CL}}$ and $\Pi_{\mathsf{DKG\text{-}Sig}}$ as ideal functionalities (in fact, they are securely realized in [28] and [19] respectively). After receiving the ECDSA public key X from $\mathcal{F}_{\mathsf{ECDSA}}$ $\mathcal{S}$ simulates

$\mathit{\Pi}_{\mathsf{DKG\text{-}CL}}$ and $\mathit{\Pi}_{\mathsf{DKG\text{-}Sig}}$, obtaining $\{(\mathsf{sk}_i, x_i)\}_{i \in I}$ and all public keys $\{(\mathsf{pk}_i, X_i)\}_{i \in I}$. For every $j \in J \backslash \{j^*\}$, $\mathcal{S}$ generates random shares $\{(\mathsf{sk}_j, x_j)\}_{j \in J \backslash \{j^*\}}$ for all honest parties except P_{j^*}, and computes their public keys $\{(\mathsf{pk}_j, X_j)\}_{j \in J \backslash \{j^*\}}$. The shared public key X_{j^*} in $\mathit{\Pi}_{\mathsf{DKG\text{-}Sig}}$ (resp. shared public key pk_{j^*} in $\mathit{\Pi}_{\mathsf{DKG\text{-}CL}}$) can be computed from the inverse of the aggregation algorithm. The correctness of the simulation has been addressed in [28] and [19], respectively. We remark that $\mathcal{S}$ does not hold the secret keys x_{j^*} and sk_{j^*} for P_{j^*}, making the simulation in signing phase challenging.

Signing Phase. $\mathcal{S}$ simulates the signing protocol as follows: Upon receiving $(\mathsf{sign}, \mathsf{sid}, m)$, it forwards this message to $\mathcal{F}_{\mathsf{ECDSA}}$ and obtains the signature (r, s) in return. $\mathcal{S}$ then computes point R using r as its x-coordinate, and proceeds to invoke $\mathcal{A}$ in the protocol execution. In the simulation of Round 1, $\mathcal{S}$ works as follows:

1. $\mathcal{S}$ samples $\phi_j \leftarrow\!\!\$\, \mathbb{Z}_q$ randomly and computes the ciphertext $\bar{\phi_j} = \mathsf{t\text{-}CL.Enc}(\mathsf{pk}, \phi_j)$ for all $j \in J$.
2. $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{zk}}^{\mathcal{R}_{\mathsf{cl\text{-}enc}}}$ sending $(\mathsf{proof}, \mathsf{sid}, \bar{\phi_j})$ to $\mathcal{A}$ for all $j \in J$. In addition, $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{dl}}}$ sending $(\mathsf{proof\text{-}receipt}, \mathsf{sid}, j)$ to $\mathcal{A}$ for all $j \in J$.
3. $\mathcal{S}$ receives $(\mathsf{prove}, \mathsf{sid}, i, \bar{\phi_i}, \phi_i)$ and $(\mathsf{com\text{-}prove}, \mathsf{sid}, i, R_i, k_i)$ as sent by $\mathcal{A}$ to $\mathcal{F}_{\mathsf{zk}}^{\mathcal{R}_{\mathsf{cl\text{-}enc}}}$ and $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{dl}}}$ for every corrupted party $i \in I$. $\mathcal{S}$ ignores any message with an illegal value or incorrect format.

For $j \in J \backslash \{j^*\}$, $\mathcal{S}$ samples $k_j \leftarrow\!\!\$\, \mathbb{Z}_q$ randomly and sets $R_j = g^{k_j}$. For $j = j^*$, $\mathcal{S}$ sets $R_{j^*} = R / \left(\prod_{i \in I} R_i \prod_{j \in J \backslash \{j^*\}} R_j\right)$.

In the simulation of Round 2, $\mathcal{S}$ works as follows:

1. $\mathcal{S}$ receives the $(\mathsf{decom\text{-}proof}, \mathsf{sid}, R_i, 1)$, and $(\overline{\phi x_i}, \pi^0_{\mathsf{dl\text{-}cl}\, i})$ and $(\overline{\phi k_i}, \pi^1_{\mathsf{dl\text{-}cl}\, i})$ as sent by $\mathcal{A}$ on behave of every corrupted party $i \in I$. $\mathcal{S}$ ignores any message with an illegal value or incorrect format.
2. $\mathcal{S}$ uses homomorphism to compute $\bar{\phi} = \bigoplus_{\underline{u \in I \cup J}} \bar{\phi_u}$ for all $j \in J$.
3. For $j \in J \backslash \{j^*\}$, $\mathcal{S}$ computes the sharing $\overline{\phi x_j} = \underline{\bar{\phi} \odot \ell_i^T} x_j$ and $\overline{\phi k_j} = \bar{\phi} \odot k_j$, then generates proofs $\pi^0_{\mathsf{dl\text{-}cl}\, j} \leftarrow \mathsf{Prove}_{\mathsf{dl\text{-}cl}}(X_j, \bar{\phi}, \overline{\phi x_j}; x_j)$ and $\pi^1_{\mathsf{dl\text{-}cl}\, j} \leftarrow \mathsf{Prove}_{\mathsf{dl\text{-}cl}}(R_j, \bar{\phi}, \overline{\phi k_j}; k_j)$.
4. For $j = j^*$, $\mathcal{S}$ first samples $\gamma \leftarrow\!\!\$\, \mathbb{Z}_q$ and sets $\gamma := \phi k$, which implies $s\gamma = \phi H(m) + \phi r x$ where s is obtained from $\mathcal{F}_{\mathsf{ECDSA}}$. Using the public key pk, $\mathcal{S}$ encrypts γ and $s\gamma$ to obtain $\overline{\gamma} \leftarrow \mathsf{t\text{-}CL.Enc}(\mathsf{pk}, \gamma)$ and $\overline{s\gamma} \leftarrow \mathsf{t\text{-}CL.Enc}(\mathsf{pk}, s\gamma)$ respectively. Under the homomorphic property of CL encryption, $\mathcal{S}$ computes:

$$\alpha := \overline{\gamma - \sum_{j \in J \backslash \{j^*\}} \phi k_j - \sum_{i \in I} \phi k_i}$$

$$\beta := \overline{\frac{s\gamma - \phi H(m)}{r} - \sum_{j \in J \backslash \{j^*\}} \phi x_j - \sum_{i \in I} \phi x_i}$$

$\mathcal{S}$ simulates the zero-knowledge proofs for P_{j^*}:

$$\pi^0_{\mathsf{dl\text{-}cl}\, j^*} \leftarrow \mathsf{Sim}_{\mathsf{dl\text{-}cl}}(X_{j^*}, \bar{\phi}, \alpha), \qquad \pi^1_{\mathsf{dl\text{-}cl}\, j^*} \leftarrow \mathsf{Sim}_{\mathsf{dl\text{-}cl}}(R_{j^*}, \bar{\phi}, \beta)$$

5. $\mathcal{S}$ sends $(\mathsf{decom\text{-}proof}, \mathsf{sid}, R_j, 1)$ to $\mathcal{A}$, and $(\overline{\phi x_j}, \pi^0_{\mathsf{dl\text{-}cl}j})$ and $(\overline{\phi k_j}, \pi^1_{\mathsf{dl\text{-}cl}j})$ to $\mathcal{A}$ on behave of P_j for all $j \in J$.

In Round 3, parties broadcast their partial decryptions of c^0 and c^1 with corresponding zero-knowledge proofs. To simulate P_{j^*}'s partial decryption $pc^0_{j^*}$ of c^0, the simulator $\mathcal{S}$ computes $(c^0)^{\Delta F(j^*)}$, where $F(u) = \Delta\mathsf{sk} + \sum_{d=0}^{t} a_d u^d$ is the polynomial from $\Pi_{\mathsf{DKG\text{-}CL}}$. The proof $\pi^0_{\mathsf{part\text{-}dec}j^*}$ is generated using zero-knoweledge simulator $\mathsf{Sim}_{\mathsf{part\text{-}dec}}(\mathsf{pk}, (c^0)^{\Delta}, pc^0_{j^*})$ without knowledge of sk_{j^*}. The same process applies to $(pc^1_{j^*}, \pi^1_{\mathsf{part\text{-}dec}j^*})$. Having acquired all necessary values in $\mathcal{A}$'s view, $\mathcal{S}$ is able to simulate for all honest parties $j \in J$. The simulation ensures the signature (r, s) from $\mathcal{F}_{\mathsf{ECDSA}}$ is output and verifies successfully.

The difference between a real execution and simulation lies in the generation of R_{j^*}, $\overline{\phi x_{j^*}}$, $\overline{\phi k_{j^*}}$, and the simulated proofs $\pi^b_{\mathsf{part\text{-}dec}j^*}$ and $\pi^b_{\mathsf{dl\text{-}cl}j^*}$ for $b \in \{0, 1\}$. In the simulation, R_{j^*} is computed as $R / \left(\prod_{i \in I} R_i \prod_{j \in J \setminus j^*} R_j\right)$, while in the real execution $R_{j^*} = g^{k_{j^*}}$ where $k_{j^*} \leftarrow\!\!\$\ \mathbb{Z}_q$. These distributions are identical since $\mathcal{F}_{\mathsf{ECDSA}}$ samples k uniformly from $\mathbb{Z}_q$ and computes $R = g^k$. The simulation uses α and β to simulate $\overline{\phi x_{j^*}}$ and $\overline{\phi k_{j^*}}$ respectively. The t-IND-CPA security of threshold CL encryption ensures that α and β do not leak any information about the plaintext, making them computationally indistinguishable from the real execution. Furthermore, the zero-knowledge property of proof systems for $\mathcal{R}_{\mathsf{part\text{-}dec}}$ and $\mathcal{R}_{\mathsf{dl\text{-}cl}}$ ensures that $\pi^b_{\mathsf{part\text{-}dec}j^*}$ and $\pi^b_{\mathsf{dl\text{-}cl}j^*}$ are computationally indistinguishable from $\mathsf{Prove}_{\mathsf{part\text{-}dec}}(\cdot)$ and $\mathsf{Prove}_{\mathsf{dl\text{-}cl}}(\cdot)$ respectively.

Consequently, the view of adversary $\mathcal{A}$ in the real execution is computationally indistinguishable from that in the simulation. That is, any PPT adversary corrupting at most t parties has only a negligible advantage in distinguishing between the real execution and the simulated execution by $\mathcal{S}$.

4 Threshold ECDSA with Robustness: TECDSA-Robust

Robustness means that with a sufficient number of honest parties, the protocol is guaranteed to output the requested signatures. The protocol Π_{TECDSA} lacks robustness because the previous k_i is the full-threshold and the absence of a single party's share will result in the failure of the signing process.

Our robust protocol requires an additional publicly verifiable secret sharing (PVSS) to generate shares of k under the t-threshold access structure, ensuring that k can be implicitly recovered when sufficient ($> t$) honest parties are present. We invoke [6]'s distributed randomness generation (DRG) to achieve robustness without increasing the number of rounds.

Distributed Randomness Generation (DRG). DRG enables n parties to cooperatively share a secret $s \in \mathbb{Z}_q$ with a threshold t and output the corresponding public value $g^s \in \mathbb{G}$. The construction of the DRG scheme Π_{DRG} in [6] follows the specifications:

Syntax. Π_{DRG} consists of three phases $(\mathsf{Setup}, \mathsf{ShareDist}, \mathsf{ShareComb})$ with the following interfaces:

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{pp}_{\mathsf{DRG}}, \{ek_i, dk_i\}_{i\in[n]})$: On input security parameter λ, it outputs public parameters and key pairs.
- $\mathsf{ShareDist}(\{ek_j\}_{j\in\mathcal{T}}, t) \to (\{C_{w_{i,j}}\}_{j\in\mathcal{T}}, \pi_{\mathsf{sh}i})$: On input encryption keys and threshold, it outputs shares and a (constant-size) consistency proof.
- $\mathsf{ShareComb}(\{C_{w_{j,i}}\}_{j\in\mathcal{T}}, dk_i) \to (w_i, W_i, \pi_{\mathsf{cl\text{-}dec\text{-}dl}})$: On input encrypted shares and decryption key, it outputs the combined secret share, public share, and a consistency proof.

Construction. The detailed construction is as follows.

- Setup: Each party P_i generates a CL encryption key pair $(ek_i, dk_i) \leftarrow \mathsf{CL.KGen}$ $(\mathsf{pp}_{\mathsf{CL}}, 1^\lambda)$, broadcasts ek_i and keeps dk_i secret.
- $\mathsf{ShareDist}$: For each party P_i does
  1. Sample random polynomial $f_i(X) \in \mathbb{Z}_q[X]$ of degree t.
  2. Compute share $w_{i,j} = f_i(j)$ and encrypt share $C_{w_{i,j}} \leftarrow \mathsf{CL.Enc}(ek_j, w_{i,j})$ for each $j \in \mathcal{T}$.
  3. Generate sharing proof $\pi_{\mathsf{sh}_i} \leftarrow \mathsf{Prove}_{\mathsf{sh}}(\{C_{x_{i,j}}\}_{j\in\mathcal{T}}; f(\cdot))$.
  4. Broadcast $(\{C_{x_{i,j}}\}_{j\in\mathcal{T}}, \pi_{\mathsf{sh}i})$ to all other parties.
- $\mathsf{ShareComb}$: Upon receiving $(\{C_{w_{j,i}}\}_{j\in\mathcal{T}}, \pi_{\mathsf{sh}_j})$ from P_j, each party P_i does
  1. Update $\mathcal{T} = \mathcal{T}\backslash\{j\}$ if verification $\mathsf{Vrfy}_{\mathsf{sh}}(\{C_{w_{j,i}}\}_{j\in\mathcal{T}}, \pi_{\mathsf{sh}_j})$ fails.
  2. Decrypt valid shares $w_{j,i} \leftarrow \mathsf{CL.Dec}(dk_i, C_{w_{j,i}})$.
  3. Combine its own secret share: $w_i = \sum_{j\in\mathcal{T}} w_{j,i} \bmod q$.
  4. Compute public share $W_i = g^{w_i} \in \mathbb{G}$.
  5. Generate decryption proof $\pi_{\mathsf{cl\text{-}dec\text{-}dl}_i} \leftarrow \mathsf{Prove}_{\mathsf{cl\text{-}dec\text{-}dl}}(C_{w_i}, W_i; w_i, dk_i)$ where $C_{w_i} = \bigoplus_{j\in\mathcal{T}} C_{w_{j,i}}$.
  6. Broadcast $(W_i, \pi_{\mathsf{cl\text{-}dec\text{-}dl}_i})$ to all other parties.

Robust Distributed Key Generation. The previous DKG for CL $\Pi_{\mathsf{DKG\text{-}CL}}$ can successfully realize the secret sharing of the CL key, as long as more than t parties are present. However, the previous DKG for ECDSA $\Pi_{\mathsf{DKG\text{-}Sig}}$ borrowed from [19] (Refer to the full version) is not robust. The reason is it utilizes VSS, instead of PVSS, the correctness of shares is not publicly verifiable.

Instead, it needs to be replaced with the above DRG, where the honest parties know the (semi)honest parties set $\mathcal{T}$ and combine a signing key share w_i correctly (Step 3 of $\mathsf{ShareComb}$). Thus, the robustness is guaranteed with DRG for generating ECDSA's key in a distributed manner.

Robust Threshold Signing. It is also a three-round scheme. The two-round DRG protocol can be executed simultaneously with the first two rounds of the non-robust threshold signing protocol. In modified Round 1, each party P_i does:

1. Choose a random $\phi_i \leftarrow\!\!\$\, \mathbb{Z}_q$ and compute the ciphertext $\bar{\phi_i} \leftarrow \mathsf{t\text{-}CL.Enc}(\mathsf{pk}, \phi_i)$.
2. Send $(\mathsf{prove}, \mathsf{sid}, i, \bar{\phi_i}, \phi_i)$ to $\mathcal{F}_{\mathsf{zk}}^{\mathcal{R}_{\mathsf{cl\text{-}enc}}}$.

3. Invoke $(\{C_{k_{i,j}}\}_{j\in\mathcal{T}}, \pi_{\mathsf{sh}i}) \leftarrow \mathsf{ShareDist}(\{ek_j\}_{j\in\mathcal{T}}, t)$.
4. Broadcast $(\bar{\phi}_i, \{C_{k_{i,j}}\}_{j\in\mathcal{T}}, \pi_{\mathsf{sh}i})$.

In modified Round 2, upon receiving $(\bar{\phi}_j, \{C_{k_{j,i}}\}_{j\in\mathcal{T}}, \pi_{\mathsf{sh}j})$ from each party P_j and $(\mathsf{proof}, \mathsf{sid}, \bar{\phi}_j)$ from $\mathcal{F}_{\mathsf{zk}}^{\mathcal{R}_{\mathsf{cl\text{-}enc}}}$, each party P_i does:

1. Verify $\mathsf{Vrfy}_{\mathsf{sh}}(\mathsf{pp}_{\mathsf{CL}}, \{ek_j\}_{j\in\mathcal{T}}, \{C_{k_{j,i}}\}_{j\in\mathcal{T}}, \pi_{\mathsf{sh}j})$ for all $j \neq i$. Update $\mathcal{T} = \mathcal{T} \setminus \{j\}$ if the verification fails.
2. Invoke $(k_i, R_i, \pi_{\mathsf{cl\text{-}dec\text{-}dl}i}) \leftarrow \mathsf{ShareComb}(\{C_{k_{j,i}}\}_{j\in\mathcal{T}}, dk_i)$.
3. Use homomorphism to compute $\bar{\phi} = \bigoplus_{j\in\mathcal{T}} \bar{\phi}_j$.
4. Compute the sharing $\overline{\phi x_i} = \bar{\phi} \odot x_i$ and generate a zero-knowledge proof $\pi^0_{\mathsf{dl\text{-}cl}i} \leftarrow \mathsf{Prove}_{\mathsf{dl\text{-}cl}}(X_i, \bar{\phi}, \overline{\phi x_i}; x_i)$.
5. Compute the sharing $\overline{\phi k_i} = \bar{\phi} \odot k_i$ and generate a zero-knowledge proof $\pi^1_{\mathsf{dl\text{-}cl}i} \leftarrow \mathsf{Prove}_{\mathsf{dl\text{-}cl}}(R_i, \bar{\phi}, \overline{\phi k_i}; k_i)$.
6. Broadcast $(\overline{\phi x_i}, \pi^0_{\mathsf{dl\text{-}cl}i})$, $(\overline{\phi k_i}, \pi^1_{\mathsf{dl\text{-}cl}i})$ and $(R_i, \pi_{\mathsf{cl\text{-}dec\text{-}dl}i})$.

In modified Round 3, each party P_i receives $(\overline{\phi x_j}, \pi^0_{\mathsf{dl\text{-}cl}j})$, $(\overline{\phi k_i}, \pi^1_{\mathsf{dl\text{-}cl}j})$ and $(R_j, \pi_{\mathsf{cl\text{-}dec\text{-}dl}j})$ from each party P_j. The differences from the non-robust protocol are as follows.

1. Each party P_i verifies $\pi^0_{\mathsf{dl\text{-}cl}j}, \pi^1_{\mathsf{dl\text{-}cl}j}, \pi_{\mathsf{cl\text{-}dec\text{-}dl}j}$ and update $\mathcal{T} = \mathcal{T} \setminus \{j\}$ if the verification fails.
2. Compute $R = \prod_{j\in\mathcal{T}} \ell_j^{\mathcal{T}} R_j$ and sets r_x as the x-coordinate of R.
3. Compute $c^0 = \bigoplus_{j\in\mathcal{T}} \overline{\phi k_j} \odot \ell_j^{\mathcal{T}}$ and $\overline{\phi x r_x} = \bigoplus_{j\in\mathcal{T}} \overline{\phi x_j} \odot (\ell_j^{\mathcal{T}} \cdot r_x)$.

Analysis of Robustness. This protocol ensures that both k_i and x_i follow a t-threshold access structure. As long as there are at least $t+1$ (semi-)honest parties, R and c^0 are generated using the same k, while X and $\overline{\phi x r_x}$ are generated using the same x. Consequently, the final produced signature is valid.

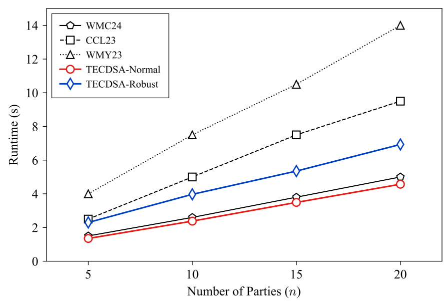
Theorem 2. *Assume that the parties $P_1, P_2, \ldots, P_n$ are invoked with the same $\big(pp_{CL}, pp_{Sig}, n, t, \mathcal{T}\big)$, where $\mathcal{T} \subset [n]$ contains at most t corrupt parties. If t-CL is simulation secure and t-IND-CPA secure, our robust threshold signature protocol securely realizes functionality $\mathcal{F}_{\mathsf{ECDSA}}$ (defined in Fig. 1) in the presence of a malicious static adversary $\mathcal{A}$ in the $(\mathcal{F}^{\mathcal{R}}_{com\text{-}zk}, \mathcal{F}^{\mathcal{R}}_{zk})$-hybrid model.*

Proof Sketch. This proof is very similar to the proof of Theorem 1. Due to the limited space, we only present the differences here.

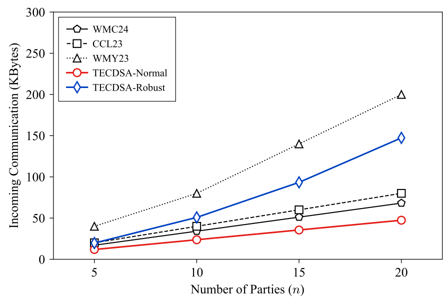
In the key generation phase, after receiving the ECDSA public key X from $\mathcal{F}_{\mathsf{ECDSA}}$, $\mathcal{S}$ simulates the DRG, making it output X as ECDSA public key and obtaining the adversary's share $\{x_i\}_{i\in I}$. In the signing phase, after receiving R from $\mathcal{F}_{\mathsf{ECDSA}}$ $\mathcal{S}$ also simulates the DRG, making it output R as the public nonce and obtaining the adversary's $\{k_i\}_{i\in I}$. The concrete strategies of DRG's simulation can be found in [6].

Signing Protocols	(1, 2)-threshold		(2, 3)-threshold		(4, 5)-threshold	
	Time	Out.Comm	Time	Out.Comm	Time	Out.Comm
CCL23 [8]	1012	6553	1518	10752	2531	19148
WMY23 [29]	1730	7680	2595	7850	4328	8192
WMC24 [28]	631	3490	938	3490	1564	3490
TECDSA-Normal	540	2426	810	2426	1350	2426
TECDSA-Robust	991	3330	1284	3564	1811	4032

Table 2: Runtime of total protocol (denoted by "Time", in ms) and (outgoing) communication overhead of offline phase (denoted by "Out.Comm", in *Bytes*) per party under threshold setting $t = 1, 2$ and 4.



(a) Runtime per party



(b) Incoming communication per party

Fig. 5: Runtime of total protocol and (incoming) communication overhead of offline phase under threshold setting $t = n - 1$.

5 Implement and Comparison

We implement our two schemes based on BICYCL [2], an open-source C++ library optimized for class group arithmetic operations. Our implementation is available at Github [3]. Then we compare our schemes with existing protocols [8], [29], and [28] in Fig. 5 and Table 2, measuring communication costs and computation time per party averaged over 100 runs.

All experiments are conducted on a virtual machine running Ubuntu 20.04 with an AMD Ryzen 7 PRO 4750U 1.70 GHz CPU and 8 GB RAM. For security configuration, we match the 128-bit security level in [28]. Therefore, we choose the 256-bit plaintext space and 1827-bit discriminant Δ_K for the class group, using secp256k1 curve and SHA-3 hash function in the threshold ECDSA scheme.

To better illustrate the difference, we present incoming communication in Fig. 5 rather than the constant outgoing communication in Table 1. Roughly

[3] `https://github.com/Jiangjiang-jiang/Three-Round-Multiparty-ECDSA` [main, robust-version]

speaking, the incoming communication is t times the outgoing communication when $t + 1$ parties are involved.

The results in Fig. 5 show that among all protocols under comparison, our basic scheme (without robustness) achieves the fewest rounds of interactions, fastest running time, and the least offline communication overhead.

Our robust scheme reduces execution time by approximately 50% and offline communication by approximately 23% compared to WMY23 [29] when $n = 20$. Additionally, it eliminates one round at the expense of increased computation and communication compared to WMC24 [28]. We note that this additional cost remains modest in small-scale settings ($n = 2, 3, 5$) as shown in Table 2. With one fewer round than WMC24 [28], our protocol has the potential to achieve greater efficiency in network environments where communication latency is a significant factor.

For comparison, while DKLs24 [17] is computationally efficient, it incurs significant communication overhead. At $t = 5$, each participant transmits 248.5 KB, which surpasses the communication costs of all the schemes listed above.

Acknowledgments. This research was supported by the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant.

References

1. Abram, D., Nof, A., Orlandi, C., Scholl, P., Shlomovits, O.: Low-bandwidth threshold ecdsa via pseudorandom correlation generators. In: 2022 IEEE Symposium on Security and Privacy (SP). pp. 2554–2572. IEEE (2022)
2. Bouvier, C., Castagnos, G., Imbert, L., Laguillaumie, F.: I want to ride my bicycl: Bicycl implements cryptography in class groups. Journal of Cryptology **36**(3), 17 (2023)
3. Brandão, L.T., Peralta, R.: Nist first call for multi-party threshold schemes. `https://nvlpubs.nist.gov/nistpubs/ir/2023/NIST.IR.8214C.ipd.pdf` (2023)
4. Braun, L., Damgård, I., Orlandi, C.: Secure multiparty computation from threshold encryption based on class groups. In: Annual International Cryptology Conference. pp. 613–645. Springer (2023)
5. Canetti, R., Gennaro, R., Goldfeder, S., Makriyannis, N., Peled, U.: Uc non-interactive, proactive, threshold ecdsa with identifiable aborts. In: ACM CCS 2020. pp. 1769–1787 (2020)
6. Cascudo, I., David, B.: Publicly verifiable secret sharing over class groups and applications to dkg and yoso. In: ASIACRYPT 2024. pp. 216–248. Springer (2024)
7. Castagnos, G., Catalano, D., Laguillaumie, F., Savasta, F., Tucker, I.: Two-party ecdsa from hash proof systems and efficient instantiations. In: CRYPTO 2019. pp. 191–221. Springer (2019)
8. Castagnos, G., Catalano, D., Laguillaumie, F., Savasta, F., Tucker, I.: Bandwidth-efficient threshold ec-dsa revisited: Online/offline extensions, identifiable aborts proactive and adaptive security. Theoretical Computer Science **939**, 78–104 (2023)
9. Castagnos, G., Laguillaumie, F.: Linearly homomorphic encryption from ddh. In: CT-RSA. pp. 487–505. Springer (2015)
10. Castagnos, G., Laguillaumie, F., Tucker, I.: Practical fully secure unrestricted inner product functional encryption modulo p. In: ASIACRYPT. pp. 733–764. Springer (2018)

11. Cramer, R.: Modular design of secure yet practical cryptographic protocols. Ph. D.-thesis, CWI and U. of Amsterdam **2** (1996)
12. Deng, Y., Ma, S., Zhang, X., Wang, H., Song, X., Xie, X.: Promise σ-protocol: How to construct efficient threshold ecdsa from encryptions based on class groups. In: ASIACRYPT. pp. 557–586. Springer (2021)
13. Desmedt, Y.: Society and group oriented cryptography: A new concept. In: ASIACRYPT. pp. 120–127. Springer (1987)
14. Desmedt, Y.: Threshold cryptosystems. In: International Workshop on the Theory and Application of Cryptographic Techniques. pp. 1–14. Springer (1992)
15. Doerner, J., Kondi, Y., Lee, E., Shelat, A.: Secure two-party threshold ecdsa from ecdsa assumptions. In: 2018 IEEE S& P. pp. 980–997. IEEE (2018)
16. Doerner, J., Kondi, Y., Lee, E., Shelat, A.: Threshold ecdsa from ecdsa assumptions: The multiparty case. In: 2019 IEEE S& P. pp. 1051–1066. IEEE (2019)
17. Doerner, J., Kondi, Y., Lee, E., Shelat, A.: Threshold ecdsa in three rounds. In: 2024 IEEE Symposium on Security and Privacy (SP). pp. 3053–3071. IEEE (2024)
18. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: CRYPTO. vol. 263, pp. 186–194. Springer (1986)
19. Gennaro, R., Goldfeder, S.: Fast multiparty threshold ecdsa with fast trustless setup. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 1179–1194 (2018)
20. Gilboa, N.: Two party rsa key generation. In: Annual International Cryptology Conference. pp. 116–129. Springer (1999)
21. Lindell, Y.: Fast secure two-party ecdsa signing. In: CRYPTO 2017. pp. 613–644. Springer (2017)
22. Lindell, Y.: How to simulate it–a tutorial on the simulation proof technique. Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich pp. 277–346 (2017)
23. Lindell, Y., Nof, A.: Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody. In: ACM CCS 2018. pp. 1837–1854 (2018)
24. Nakamoto, S., Bitcoin, A.: A peer-to-peer electronic cash system. Bitcoin.–URL: https://bitcoin. org/bitcoin. pdf **4**(2), 15 (2008)
25. Pedersen, T.P.: A threshold cryptosystem without a trusted party. In: EUROCRYPT'91. pp. 522–526. Springer (1991)
26. Schnorr, C.P.: Efficient signature generation by smart cards. Journal of cryptology **4**, 161–174 (1991)
27. Tang, G., Han, S., Lin, L., Wei, C., Yan, Y.: Batch range proof: How to make threshold ecdsa more efficient. In: ACM CCS 2024. pp. 4256–4270 (2024)
28. Wong, H.W., Ma, J.P., Chow, S.S.: Secure multiparty computation of threshold signatures made more efficient. In: NDSS 2024 (2024)
29. Wong, H.W., Ma, J.P., Yin, H.H., Chow, S.S.: Real threshold ecdsa. In: NDSS (2023)
30. Wood, G.: A secure decentralised generalised transaction ledger. Ethereum project yellow paper **151**, 1–32 (2014)
31. Xie, Z., Liu, M., Xue, H., Au, M.H., Deng, R.H., Yiu, S.M.: Direct range proofs for paillier cryptosystem and their applications. In: ACM CCS. pp. 899–913 (2024)
32. Xue, H., Au, M.H., Liu, M., Chan, K.Y., Cui, H., Xie, X., Yuen, T.H., Zhang, C.: Efficient multiplicative-to-additive function from joye-libert cryptosystem and its application to threshold ecdsa. In: ACM CCS. pp. 2974–2988 (2023)
33. Xue, H., Au, M.H., Xie, X., Yuen, T.H., Cui, H.: Efficient online-friendly two-party ecdsa signature. In: ACM CCS. pp. 558–573 (2021)

34. Yuen, T.H., Cui, H., Xie, X.: Compact zero-knowledge proofs for threshold ecdsa with trustless setup. In: PKC. pp. 481–511. Springer (2021)

A Zero-Knowledge Proof

This section details the Σ-protocols for relations $\mathcal{R}_{\mathsf{dl}}$, $\mathcal{R}_{\mathsf{dl\text{-}cl}}$, $\mathcal{R}_{\mathsf{part\text{-}dec}}$, $\mathcal{R}_{\mathsf{sh}}$, and $\mathcal{R}_{\mathsf{cl\text{-}dec\text{-}dl}}$ from Section 2.3, along with $\mathcal{R}_{\mathsf{BInt}}$ and $\mathcal{R}_{\mathsf{cl\text{-}dec}}$ for DKG of threshold CL encryption (Appendix B.1).

Denote pp_1 be the public parameters $(\mathbb{G}, g, q)$ over class groups, and pp_2 be the public parameters $(\widetilde{s}, f, G^q, g_q, \widehat{G}, F, q)$ over Elliptic curve groups. Let $\mathcal{D} = 2^\lambda q\widetilde{s}$ be the upper bound of $\mathcal{D}_q$, and λ_d be a statistical parameter.

The Σ-protocol for relation $\mathcal{R}_{\mathsf{dl}}$ follows the standard discrete logarithm zero-knowledge protocol under parameters pp_1, while the protocol instantiated by $\mathcal{R}_{\mathsf{sh}}$ refers to [6] under parameters pp_2.

A.1 Proof of CL Encryption

Given parameters pp_2, $c = (c_1, c_2)$, the Σ-protocol for the relation:

$$\mathcal{R}_{\mathsf{cl\text{-}enc}} = \left\{ (\mathsf{pk}, c; m, r) \;\middle|\; r \in \mathcal{D}_q,\ c_1 = \left(g_q^{\Delta^2}\right)^r,\ c_2 = f^m \mathsf{pk}^r \right\}$$

between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ is as follows.

1. $\mathcal{P}$ chooses $r_1 \hookleftarrow\$\ \mathbb{Z}_q$, $r_2 \hookleftarrow\$\ [0, 2^{\lambda+\lambda_d}\mathcal{D}]$, and computes $t_1 = g_q^{r_2}$, $t_2 = f^{r_1}\mathsf{pk}^{r_2}$. Next, $\mathcal{P}$ sends (t_1, t_2) to $\mathcal{V}$.
2. $\mathcal{V}$ selects $e \hookleftarrow\$\ [0, 2^\lambda]$ and sends it to $\mathcal{P}$.
3. $\mathcal{P}$ computes $z_1 = r_1 + e \cdot m$ (over $\mathbb{Z}_q$), $z_2 = r_2 + e \cdot r$ (over $\mathbb{Z}$), then sends (z_1, z_2) to $\mathcal{V}$.
4. $\mathcal{V}$ accepts the proof if all conditions hold:
   - $z_1 \in \mathbb{Z}_q$ and $z_2 \in [0, (2^\lambda \mathcal{D})(2^{\lambda_d} + 1)]$
   - $g_q^{z_2} = t_1 \cdot {c_1}^e$
   - $f^{z_1}\mathsf{pk}^{z_2} = t_2 \cdot {c_2}^e$

A.2 Proof of Multiplying CL Ciphertext with Discrete Logarithm

Given parameters $(\mathsf{pp}_1, \mathsf{pp}_2)$, $c_0 = (c_{01}, c_{02})$, $c_1 = (c_{11}, c_{12})$ the Σ-protocol for the relation:

$$\mathcal{R}_{\mathsf{dl\text{-}cl}} = \{(X, c_0, c_1; x) \mid X = g^x, c_{01} = {c_{11}}^x, c_{02} = {c_{12}}^x\}$$

between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ is as follows.

1. $\mathcal{P}$ chooses $r_1 \hookleftarrow\$\ \mathbb{Z}_q, r_2 \hookleftarrow\$\ [0, 2^{\lambda+\lambda_d}q]$, and computes , $S = g^{r_1}, t_1 = {c_{11}}^{r_1}, t_2 = {c_{12}}^{r_2}$. Next, $\mathcal{P}$ sends (S, t_1, t_2) to $\mathcal{V}$.
2. $\mathcal{V}$ selects $e \hookleftarrow\$\ [0, 2^\lambda]$ and sends it to $\mathcal{P}$.

3. $\mathcal{P}$ computes $z_1 = r_1 + e \cdot x$ (over $\mathbb{Z}_q$), $z_2 = r_2 + e \cdot x$ (over $\mathbb{Z}$), then sends (z_1, z_2) to $\mathcal{V}$.
4. $\mathcal{V}$ accepts the proof if all conditions hold:
   - $z_1 \in \mathbb{Z}_q$ and $z_2 \in [0, (2^{\lambda}\mathcal{D})(2^{\lambda_d} + 1)]$
   - $g^{z_1} = SX^e$
   - ${c_{01}}^{z_2} = t_1 \cdot {c_{11}}^e$
   - ${c_{02}}^{z_2} = t_2 \cdot {c_{12}}^e$

A.3 Proof of Correct Partial Decryption

Given parameters pp_2, c_1 is the first part of c, of which pc_i is the partial decryption, the Σ-protocol for the relation:

$$\mathcal{R}_{\textsf{part-dec}} = \left\{ (\mathsf{pk}_i, c_1^{\Delta}, pc_i; \mathsf{sk}_i) \;\middle|\; \mathsf{pk}_i = (g_q^{\Delta})^{\mathsf{sk}_i}, pc_i = (c_1^{\Delta})^{\mathsf{sk}_i} \right\}$$

between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ is as follows.

1. $\mathcal{P}$ chooses $r \leftarrow\!\$\, [0, 2^{\lambda+\lambda_d}\mathcal{D}]$, and computes $t_1 = g_q^r, t_2 = {c_1}^r$. Next, $\mathcal{P}$ sends (t_1, t_2) to $\mathcal{V}$.
2. $\mathcal{V}$ selects $e \leftarrow\!\$\, [0, 2^{\lambda}]$ and sends it to $\mathcal{P}$.
3. $\mathcal{P}$ computes $z = r + e \cdot \mathsf{sk}_i$ (over $\mathbb{Z}$), then sends z to $\mathcal{V}$.
4. $\mathcal{V}$ accepts the proof if all conditions hold:
   - $z \in [0, (2^{\lambda}\mathcal{D})(2^{\lambda_d} + 1)]$
   - $g_q^z = t_1 \cdot \mathsf{pk}_i^e$
   - ${c_1}^z = t_2 \cdot {pc_i}^e$

A.4 Proof of Validity of Encrypted Shares

Given parameters pp_2, the Σ-protocol for the relation:

$$\mathcal{R}_{\mathsf{sh}} = \left\{ (\{C_{f(i)}\}_{i \in \mathcal{T}}, f(u)) \;\middle|\; \begin{array}{l} f(u) \in \mathbb{Z}_q[u]_{\leq t}, \\ \forall i \in \mathcal{T} : C_{f(i)} = \mathsf{CL.Enc}(ek_i, f(i)) \end{array} \right\}$$

The instantiation of Σ-protocol for $\mathcal{R}_{\mathsf{sh}}$ refers to the [6, Figure 4].

A.5 Proof of CL Decrypted Plaintext with Discrete Logarithm

Given parameters $(\mathsf{pp}_1, \mathsf{pp}_2)$, $C_a = (c_1, c_2)$, the Σ-protocol for the relation:

$$\mathcal{R}_{\textsf{cl-dec-dl}} = \{ (C_a, A; a, dk) \mid a \in \mathbb{Z}_q,\, dk \in \mathcal{D}_q,\, A = g^a,\, a = \mathsf{CL.Dec}(dk, C_a) \}$$

between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ is as follows.

1. $\mathcal{P}$ chooses $r_1 \leftarrow\!\$\, \mathbb{Z}_q, r_2 \leftarrow\!\$\, [0, 2^{\lambda+\lambda_d}\mathcal{D}]$, and computes $S = g^{r_1}, t_1 = f^{r_1} c_1^{r_2}, t_2 = g_q^{r_2}$. Next, $\mathcal{P}$ sends (S, t_1, t_2) to $\mathcal{V}$.
2. $\mathcal{V}$ selects $e \leftarrow\!\$\, [0, 2^{\lambda}]$ and sends it to $\mathcal{P}$.

3. $\mathcal{P}$ computes $z_1 = r_1 + e \cdot a$ (over $\mathbb{Z}_q$), $z_2 = r_2 + e \cdot dk$ (over $\mathbb{Z}$), then sends (z_1, z_2) to $\mathcal{V}$.
4. $\mathcal{V}$ accepts the proof if all conditions hold:
   - $z_1 \in \mathbb{Z}_q$ and $z_2 \in [0, (2^{\lambda}\mathcal{D})(2^{\lambda_d} + 1)]$
   - $g_q^{z_1} = SA^e$
   - $g_q^{z_2} = t_2 \cdot ek^e$
   - $f^{z_1} c_1^{z_2} = t_1 \cdot c_2^e$
   - ${c_1}^z = t_2 \cdot {pc_i}^e$

A.6 Proof of Encryption and Commitment of Big Integers

We define the relation $\mathcal{R}_{\mathsf{BInt}}$ for $\Pi_{\mathsf{DKG\text{-}CL}}$ shown in Appendix B.1. The relation verifies that PC is a well-formed Pedersen commitment over class group of a big integer $x \in \mathcal{D}_q$, $EC = (c_1, c_2)$ is the ElGamal ciphertext over class group encrypting g_q^x with the q-ary expression of $x = \left(\sum_\ell q^\ell x_\ell\right)$, where x_ℓ is encrypted in C_ℓ.

$$\mathcal{R}_{\mathsf{BInt}} = \left\{ (PC, \{C_\ell\}_\ell, EC, ek; \{x_\ell\}_\ell, x', r) \;\middle|\; \begin{array}{l} PC = g_q^{x'} \prod_\ell (h^{q^\ell})^{x_\ell}, \\ \forall \ell \in [0, \mathsf{len} - 1] : \{C_\ell \leftarrow \mathsf{CL.Enc}(ek, x_\ell)\}_\ell, \\ r \leftarrow\!\!\$\; \mathcal{D}_q, c_1 = g_q^r, c_2 = ek^r \prod_\ell ({g_q}^{q^\ell})^{x_\ell} \end{array} \right\}$$

The instantiation of Σ-protocol for $\mathcal{R}_{\mathsf{BInt}}$ refers to [28].

A.7 Proof of Decryption of CL Encryption for Group Elements and Discrete Logarithm over Class Groups

We define the relation $\mathcal{R}_{\mathsf{cl\text{-}dec}}$ for $\Pi_{\mathsf{DKG\text{-}CL}}$ shown in Appendix B.1. The relation verifies that x such that $X = \left(g_q^{\Delta}\right)^x$ and g_q^x is decrypted from ciphertext $c = (c_1, c_2)$ under decrypted key dk.

$$\mathcal{R}_{\mathsf{cl\text{-}dec}} = \left\{ \left(X, c, ek, g_q^{\Delta}; dk, x\right) \;\middle|\; X = \left(g_q^{\Delta}\right)^x, c_2 = g_q^x c_1^{dk}, ek = g_q^{dk} \right\}$$

The instantiation of Σ-protocol for $\mathcal{R}_{\mathsf{cl\text{-}dec}}$ refers to [28].

B DKG for Key Generation

B.1 DKG for CL $\Pi_{\mathsf{DKG\text{-}CL}}$.

We borrow the techniques from [28] to implement CL's distributed key generation, which can be used in both TECDSA-Normal and TECDSA-Robust schemes. In fact, for the scheme TECDSA-Normal that does not consider robustness, we could also use the DKG for CL from [4]. The difference between the two protocols lies in the ability for public verification: [28]'s protocol allows each party to publicly verify the shares of others, whereas [4]'s protocol does not.

Round 1. Each party P_i acts as follows:

1. Sample two random coefficient vectors $\mathbf{b}_i = (b_{i0}, \dots, b_{it}) \in \left[0, 2^{\lambda_d+\lambda}\right)^{t+1}$ and $\mathbf{b}'_i = (b'_{i0}, \dots, b'_{it}) \in \left[0, 2^{\lambda_d+\lambda}\right)^{t+1}$, which defines two t-degree polynomials over $\mathbb{Z}$: $f_i(u) = b_{i0}\Delta + \sum_{k=1}^{t} b_{ik}u^k$ and $f'_i(u) = b'_{i0}\Delta + \sum_{k=1}^{t} b'_{ik}u^k$.
2. For each $j \in \mathcal{T}$, compute the share $f_{ij} = f_i(j), f'_{ij} = f_i(j)'$ and the Pedersen commitment over class group: $PC_{f_{ij}} = h^{f_{ij}} g_q^{f'_{ij}}$ where $h \in G^q$ is public parameter.
3. Compute ElGamal ciphertext over class group $EC_{f_{ij}} = (g_q^r, g_q^{f_{ij}\Delta} ek_j^r)$ where $r \leftarrow\$ \mathcal{D}_q$.
4. Write f_{ij} in q-ary: $f_{ij} = \sum_{\ell=0}^{\mathsf{len}-1} q^\ell f_{ij\ell}$, with $\mathsf{len} = \lceil \frac{\log(n^t \cdot 2^{\lambda_d+\lambda})}{\log q} \rceil$. For each $\ell \in [0, \mathsf{len}-1]$, compute the CL ciphertext $C_{f_{ij\ell}} \leftarrow \mathsf{CL.Enc}(ek_j, f_{ij\ell})$.
5. Generate $\pi_{\mathsf{BInt}ij} \leftarrow \mathsf{Prove}_{\mathsf{BInt}}(PC_{f_{ij}}, \{C_{f_{ij\ell}}\}_\ell, EC_{f_{ij}}, ek_j; \{f_{ij\ell}\}_\ell, f'_{ij}, r)$
6. Broadcast $\{PC_{f_{ij}}, \{C_{f_{ij\ell}}\}_\ell, EC_{f_{ij}}, \pi_{\mathsf{BInt}ij}\}_{j\in\mathcal{T}}$.

Round 2. Upon receiving the proof $\pi_{\mathsf{BInt}ij}$ from P_j, if it is not valid, update $\mathcal{T} = \mathcal{T} \setminus \{j\}$. Then P_i does

1. For each $j \in \mathcal{T}$, compute $f_{ji} = \sum_{\ell=0}^{\mathsf{len}-1} q^\ell \mathsf{CL.Dec}(dk_i, C_{f_{ij\ell}})$, assemble $\mathsf{sk}_i = \sum_{j\in\mathcal{T}} f_{ji}$. Compute $\mathsf{pk}_i = g_q^{\mathsf{sk}_i\Delta}$, and parse $EC_{f_{ji}} = (c_j^0, c_j^1)$ and $EC_{\mathsf{sk}_i} = (\prod_j c_j^0, \prod_j c_j^1)$.
2. Generate $\pi_{\mathsf{cl\text{-}dec}_i} \leftarrow \mathsf{Prove}_{\mathsf{cl\text{-}dec}}(\mathsf{pk}_i, EC_{\mathsf{sk}_i}, ek_i, g_q^\Delta; dk_i, \mathsf{sk}_i)$.
3. Broadcast $(\mathsf{pk}_i, \pi_{\mathsf{cl\text{-}dec}_i})$.

Output. Upon receiving the proof $\pi_{\mathsf{cl\text{-}dec}_j}$ from P_j, if it is not valid, update $\mathcal{T} = \mathcal{T} \setminus \{j\}$. Then P_i stores the t-threshold share sk_i, public key share pk_i and others' public shares $\{\mathsf{pk}_j\}_{j\neq i}$. The global public key is $\mathsf{pk} = \prod_{j\in\mathcal{T}} \mathsf{pk}_j^{\Delta\ell_j^{\mathcal{T}}}$.

B.2 DKG for ECDSA $\Pi_{\mathsf{DKG\text{-}Sig}}$.

Protocol $\Pi_{\mathsf{DKG\text{-}Sig}}$ is roughly that in [19] for the distributed key generation of ECDSA's key pair. It builds upon [25], taking as input the public parameters $(\mathsf{pp}_{\mathsf{Sig}} = (\mathbb{G}, g, q), n, t)$, with t-threshold setting among $P_1, P_2, \dots, P_n$.

Round 1. Each party P_i acts as follows:

1. Sample a random coefficient vector $\mathbf{a}_i = (a_{i0}, \dots, a_{it}) \in \mathbb{Z}_q^{t+1}$, which defines a t-degree polynomial $f_i(u) = \sum_{k=0}^{t} a_{ik}u^k$ over the field $\mathbb{Z}_q$.
2. Compute public commitments vector $\mathbf{A}_i = (A_{it}, \dots, A_{i1}) \in \mathbb{G}^t$ where $A_{ik} = g^{a_{ik}}$ for $k \in \{1, \dots, t\}$.
3. Compute $A_{i0} = g^{a_{i0}}$ and send $(\mathsf{com\text{-}prove}, \mathsf{sid}, i, A_{i0}, a_{i0})$ to $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{dl}}}$.

Round 2. Upon receiving $(\mathsf{proof\text{-}receipt}, \mathsf{sid}, j)$ from $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{dl}}}$ for P_j, P_i does as follows:

1. Send $(\mathsf{decom\text{-}proof}, \mathsf{sid}, i)$ to $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{dl}}}$.
2. Broadcast $\mathbf{A}_i$ and send $f_i(j)$ to P_j for each $j \neq i$.

Output. Upon receiving $(\mathsf{decom\text{-}proof}, \mathsf{sid}, A_{j0}, 1)$ from $\mathcal{F}^{\mathcal{R}_{\mathsf{dl}}}_{\mathsf{com\text{-}zk}}$ for P_j, P_i verifies their shares by checking the equation $g^{f_j(i)} = \prod_{k=0}^{t} A_{jk}{}^{i^k}$, aborting if the verification fails. Otherwise, P_i follows:

1. Compute and store the t-threshold share $x_i = \sum_{j=1}^{n} f_j(i) \mod q$, which would be used for threshold ECDSA signing protocol.
2. Calculate and store the public key share $X_i = g^{x_i}$ and other parties' $X_j = \prod_{l=1}^{n} \prod_{k=0}^{t} A_{lk}{}^{j^k}$ for each $j \neq i$. The global public key is $X = \prod_{j=1}^{n} A_{j0}$.