Robust Threshold ECDSA with Online-Friendly Design in Three Rounds

Guofeng Tang*
Singapore Management University
Email: tang.guofeng789@gmail.com

Haiyang Xue
Singapore Management University
Email: haiyangxc@gmail.com

***Abstract*—Threshold signatures, especially ECDSA, enhance key protection by addressing the single-point-of-failure issue. Threshold signing can be divided into offline and online phases, based on whether the message is required. Schemes with low-cost online phases are referred to as "online-friendly". Another critical aspect of threshold ECDSA for real-world applications is robustness, which guarantees the successful completion of each signing execution whenever a threshold number t of semi-honest participants is met, even in the presence of misbehaving signatories.**

The state-of-the-art online-friendly threshold ECDSA without robustness was developed by Doerner et al. in S&P'24, requiring only three rounds. Recent work by Wong et al. in NDSS'23 (WMY$^+$23) and NDSS'24 (WMC24) achieves robustness but demands additional communication rounds (7 and 4, respectively) or incurs costly operations in the online phase, such as computations over a homomorphic encryption scheme.

This paper presents the first three-round threshold ECDSA scheme with both robustness and an online-friendly design. The online phase of our scheme relies solely on several elliptic-curve group operations, which are 2 to 3 orders of magnitude less computationally intensive than those based on linearly homomorphic encryption schemes. We implement our protocol and conduct a comprehensive comparison with WMY$^+$23 and WMC24. Benchmark results show that the online phase of our scheme is $2.5\times$ faster than that of WMY$^+$23 and hundreds of times faster than that of WMC24. Lastly, we demonstrate that our techniques can be extended to construct an online-friendly and robust three-round threshold BBS+ scheme.

1. Introduction

A (t, n) threshold signature [33] splits the signing key across n parties, such that: (1) any subset of t honest parties can produce a valid signature without reconstructing the key, and (2) any subset of fewer than t parties can not [43], [53]. Threshold signatures provide enhanced key protection by mitigating the risk of single points of failure, making them highly valuable in both blockchain applications and academic research. Their significance can be recognized from NIST's upcoming standardization efforts on Multi-Party Threshold Cryptography [16].

∗. Part of this work was done when Guofeng Tang was at Ant Group.

With the rising demand for blockchain-based cryptocurrencies, the design and deployment of threshold ECDSA have seen widespread popularity over the past decade. This includes seminal contributions by [34], [36], [41], [47], [48] and implementations by industry like Coinbase [3], Binance [1], Zengo [5], BitGo [2], and Fireblocks [4].

Cheater Identification & Robustness. The fundamental security requirement for threshold ECDSA is the ability to tolerate up to $t-1$ malicious (or corrupted) participants. Beyond this basic security, additional features are valuable in practice, such as *cheater identification* [8], [20], [24], [40] and *robustness* [58], [59].

Cheater identification (also known as *identifiable abort* if abort after identification) allows for the detection of misbehaving parties. Typically, we can design a protocol that eliminates the cheaters and restarts the signing process until a signature is produced. This will cause the round count to be related to the number of corruptions, which is unacceptable, especially in time-sensitive applications. Even worse, the signature string can be biasable by the adversary, which is an attack if it is used for leader election.

Therefore, they propose the robustness feature, which, after identifying cheaters, allows the signing process to continue and produce a valid signature as long as the remaining (semi)honest parties still meet the threshold t. It is worth noting, as analyzed in [29], that an honest majority ($n \geq 2t-1$) appears to be necessary for achieving robustness (i.e., the public key and/or signatures are not adversarially biased).

Online and Offline Signing. In general, it is highly advantageous to separate a multiparty computation (MPC) protocol into *offline* and *online* phases. During the offline phase, input-independent computations are performed and can be batched during idle CPU time. The results from the offline phase are stored for future use in the online phase, when specific inputs become available, with the goal of minimizing the online computational cost. An example is the use of Beaver triples [9] to facilitate efficient computation of multiplication gates in the online phase. The same principle applies to threshold ECDSA protocols, which can be divided into offline (pre-signing) and online phases, depending on whether the signing message is required. The goal is to make online signing efficient by shifting computationally intensive tasks to the offline phase, such as *Multiplicative-to-Additive*

(MtA) conversions. Protocols are considered *online-friendly* if the online phase requires only minimal elliptic curve (EC) group operations, as in [24], [36], [59]. Others [22], [47], [58] rely on costly operations with linearly homomorphic encryption (LHE) in the online phase. Notably, some schemes [20] are online-friendly but still require expensive operations for cheater identification.

Current Threshold ECDSAs. To motivate our work, we briefly introduce existing threshold ECDSA schemes based on different building blocks and discuss their pros and cons regarding rounds, online complexity, cheater identification, and robustness.

OT-based schemes. Doerner et al. proposed a series of threshold ECDSA schemes based on Oblivious Transfer (OT) [34], [35], [36]. These schemes have very fast running times but require substantial bandwidth (over $49.7n$ KB when n parties are involved). The most advanced among them, [36] (hereafter referred to as DKLs24), achieves three communication rounds and extremely low online costs. However, it can only achieve basic security without identifiable abort or robustness. Recently, Cohen et al. [30] added an identifiable abort to DKLs24, but it requires seven rounds of communication.

LHE-based scheme. Linearly homomorphic encryption (LHE) is another common building block of threshold ECDSA, such as Paillier encryption (e.g., [11], [41], [42], [47], [48]), Castagnos-Laguillaumie encryption (CL, e.g., [23], [24]), and Joye-Libert encryption (JL, e.g., [60]). These constructions have low bandwidth but require higher computation overhead compared to those from OT. Some LHE-based solutions enable cheater identification, as in CGG$^+$20 [20], and robustness, as in WMY$^+$23 [59]. However, CGG$^+$20 suffers from expensive online cheater identification, and WMY$^+$23 requires seven rounds to achieve robustness.

TLHE-based scheme. Inspired by [17], WMC24 [58] proposes a threshold ECDSA scheme with cheater identification and robustness by employing threshold CL encryption. It achieves a constant (sending) communication overhead for each participant. However, due to the threshold decryption operations and the associated zero-knowledge proofs (ZKPs), the scheme is not online-friendly and requires at least four communication rounds.

We provide a comparison of state-of-the-art works in each category in Table 1, evaluating them based on rounds, cheater identification (CI), robustness, and online cost. In summary, current constructions either achieve a three-round, online-friendly design but lack CI and robustness, or they provide CI and/or robustness at the expense of additional rounds and/or higher online costs. This paper is therefore motivated by the goal of designing a three-round threshold ECDSA scheme that incorporates CI, robustness, and low online cost.

Schemes	Types	Rounds	CI	R	Online Cost
DKLs24 [36]	OT	3	×	×	Optimal
CDKS24 [30]	OT	7	✓	×	2×DDH-ZKP
CGG$^+$20 [20]	LHE	4	✓	×	Enc, Paillier-ZKP
CCL$^+$23 [24]	LHE	7	✓	×	2×DDH-ZKP
WMY$^+$23 [59]	LHE	5+2*	✓	✓†	3×DDH-ZKP
WMC24 [58]	TLHE	4	✓	✓	Dec, CL-ZKP
Ours	LHE	3	✓	✓	DDH-ZKP

* "+2" indicates the extra cost for CI.
† We show its vulnerability when combining CI and R properties.

TABLE 1: **Comparisons of threshold ECDSAs, in terms of rounds, CI (cheater identification), R (robustness) and online cost.** "Optimal" indicates the online phase (OP) has no more EC group operations except for signature verification. "Enc", "Dec", "Paillier-ZKP", and "CL-ZKP" indicate encryption, decryption, Paillier or CL-related ZKPs. "DDH-ZKP" means ZKPs for DDH relation. The OPs of [24], [30], [59] require ZKPs over the EC group for relations more complex than DDH, which can be considered approximately as $2\times$ or $3\times$ DDH-ZKP.

1.1. Our Contributions

We present an online-friendly robust threshold ECDSA that requires three rounds, comparable with the state-of-the-art scheme without robustness. We also extend our technique to thresholding BBS+ [7], a crucial primitive for anonymous credential issuance.

- As shown in Table 1, current techniques for adding robustness to threshold ECDSA [58], [59] require at least one extra round to the already three-round signing phase. We successfully achieve CI and robustness without additional rounds. Moreover, the online phase of our protocol relies on only a few elliptic curve operations. Two novel techniques are developed to achieve this.
  a) Our main observation is that an *undetermined temporary secret nonce*[1] is sufficient for robust threshold ECDSA. Using this indeterminacy allows us to avoid the additional round of interaction needed to generate a determined temporary secret nonce[2], as required in [59]. Despite the undetermined nature, the signatories can still successfully assemble the signature (i.e., achieve robustness) because the value will ultimately be canceled out.
  b) Furthermore, in the online phase, we use a *simple zero-knowledge proof for the decisional Diffie-Hellman (DDH) relation*, which requires only inexpensive elliptic curve operations. This zero-knowledge proof is more cost-effective than those required in the online phases of [58]. Notably, the DDH-relation proof en-

1. An *undetermined* secret nonce is shared into additive shares (e.g., $r = \sum_{i=1}^{n} r_i$). The value assembled by one set of parties differs from that assembled by another. As malicious parties are removed, the set of signers is undetermined. Thus, the value is also undetermined.

2. A *determined* secret nonce is shared in the t-threshold form using Shamir's secret sharing. The value assembled by *any* $\geq t$ parties is determined.

Schemes	Types	Rounds	CI	R	Online Cost
DKL$^+$23 [37]	OT	3	×	×	Optimal
WMC24 [58]	TLHE	4	✓	✓	Dec, CL-ZKP
Ours	LHE	3	✓	✓	Pairings

TABLE 2: **Comparisons of threshold BBS+, in terms of rounds, CI (cheater identification), R (robustness) and online cost.** "Pairings" indicates pairing operations, which are almost free compared with those of the CL scheme.

ables the CI mechanism, allowing parties to publicly identify and exclude misbehaving parties.

Details on the techniques are given in Section 1.2.2.

- We show that *public checking* is crucial for enabling secure CI in a robust threshold signature scheme by highlighting an attack to WMY$^+$23 [59]. The vulnerability of WMY$^+$23 is due to using MtA with *private checking* (see more below). While private checking requires secret input, public checking does not rely on any secret, allowing anyone to openly validate the transmitted messages. Private checking necessitates additional steps that may lead to secret leakage to achieve the effect of open validation, as shown in our attack. Therefore, we opt for public checking. Specifically, we propose a CL-based multi-party MtA with pubic checking (c.f. Section 3.3, which is of independent interest), and show that multiple proofs used for public checking can be aggregated into a constant-size one, along with batch-proving and batch-verification.
- We provide a comprehensive implementation and comparison with WMY$^+$23 and WMC24. The results and comparisons are summarized in Table 4 and illustrated in Fig. 3. Our online phase (resp. offline phase) is approximately $2.5\times$ (resp. $1.3\times$) faster than that of WMY$^+$23. Additionally, our online computation is significantly faster than WMC24's (0.5 ms vs. 308 ms in the two-party setting). However, our overall performance outperforms WMC24 only in smaller-scale cases: specifically, our overall runtime is faster when $n \leq 12$, and our total bandwidth is lower when $n \leq 5$. Fortunately, small-scale scenarios are more common in practical applications. A detailed discussion is provided in Section 1.3.
- At last, we extend our technique to a robust threshold BBS+ signature, which faces similar challenges with robust threshold ECDSA. Threshold BBS+ is a crucial primitive to design distributed anonymous credential issuance [28], [50], [54]. Our robust threshold BBS+ also achieves robustness in three rounds, as shown in Table 2. Since our online phase only requires operations over the EC group, it is much faster than WMC24 [58], achieving 2.48 ms compared to 287 ms in the two-party setting. Similarly, our overall overhead performs better than WMC24 only when $n < 10$. The experimental results are shown in Table 5 and illustrated in Fig. 4.

1.2. Technical Overview

We first recall current constructions, analyze the challenges to achieve a three-round scheme with cheater identification (CI) and robustness, and then give our solution.

1.2.1. Backgrounds and Challenges. Let $\mathbb{G} =< G >$ be the group used for ECDSA with order q. Recall that an ECDSA signature on a message msg involves computing a public nonce $r = kG|_{x\text{-axis}}$ and $s = k^{-1} \cdot (m + r \cdot x)$, where x is the signing key, k is a secret nonce, and $m = \mathcal{H}(\mathsf{msg})$. With a **temporary secret nonce** γ, the s-component can be rewritten as

$$s = \frac{\gamma(m + r \cdot x)}{\gamma k} \bmod q. \tag{1}$$

The primary challenge in threshold ECDSA is securely computing (non-linear) s from shares of x, k, γ (denoted as (x_i, k_i, γ_i) for the i-th party) in a distributed manner.

Several protocols, including the state-of-the-art DKLs24 [36], address this issue using Multiplicative-to-Additive (MtA) functionality, which, given inputs a and b, computes and outputs α and β such that $\alpha + \beta = ab$. Thus, we begin by reviewing DKLs24 and then explaining the challenges related to cheater identification and achieving robustness.

Assume n parties generate an ECDSA public key $X = xG$ and each has a t-threshold share x_i of secret key x[3]. Assume n' participants are involved in the signing phase for any $t \leq n' \leq n$. DKLs24 has the following skeleton.

- Round 1 and 2 (offline phase): Each party randomly selects k_i and γ_i. Using MtA, the parties compute additive shares δ_i of γk, such that $\gamma k = \delta_1 + \cdots + \delta_{n'}$, and additive shares ζ_i of γx, satisfying $\gamma x = \zeta_1 + \cdots + \zeta_{n'}$. Simultaneously, they execute a standard commit-and-release mechanism to assemble the pre-signature $R = \sum_i k_i G$. All tasks are completed within two rounds.
- Round 3 (online phase): Given a message $m \in \mathbb{Z}_q$, broadcast δ_i and $\chi_i = m\gamma_i + r\zeta_i$ where $r = R|_{x\text{-axis}} \in \mathbb{Z}_q$.

They leverage the useful property of MtA to convert non-linear computations into additive shares, but the protocol lacks cheater identification (CI) and robustness due to the absence of public verifiability and fault tolerance (e.g., when some parties use incorrect values or are absent during the signing). Although DKLs24 provides an efficient MAC-style statistical check before the final round, if $\mathcal{P}_i$ uses an incorrect share γ_i^* to evaluate χ_i in the final round, honest parties will only detect cheating after verifying the final signature $(r, s = \sum_i \chi_i / \sum_i \delta_i)$. Worse, they cannot identify which party is responsible for the cheating, requiring a restart of the process until no cheating occurs.

For cheater identification. *MtA with (private or public) checking* has been suggested as a solution for identifying cheaters, as shown by WMY$^+$23 [59] and CGG$^+$20 [20]. However, only MtA with *public checking* is secure when robustness is also required.

3. We say $\{x_i\}_{i\in[n]}$ are t-threshold shares of x, if any n' such shares can reconstruct x if $n' \geq t$.

- Private checking: The works of [24] and WMY$^+$23 achieve cheater identification (CI) using MtA with private checking, where one party, Alice, can verify the correctness of another party, Bob, using her secret share. This requires Alice to identify cheaters by publicly revealing *her secret share*. While this approach is acceptable if only CI is required, as in [24], it presents a specific vulnerability if robustness (i.e., the ability to continue despite faults) is also needed, as required by WMY$^+$23. We demonstrate this vulnerability in Section 3.2. In brief, an adversary controlling $t-1$ parties could eventually reconstruct the secret signing key from Alice's revealed shares and the generated signature (under robustness requirements).
- Public checking: CGG$^+$20 [20] employed MtA with public checking for cheater identification (CI), using publicly verifiable zero-knowledge proofs (ZKPs). These ZKPs allow any party to identify a cheater. However, this approach introduces significant computational overhead, especially in the online phase, due to the use of Paillier encryption and the associated ZKPs.

In conclusion, MtA with public checking is crucial for achieving both CI and robustness simultaneously. Since OT-based MtA with public checking remains unclear (e.g., [36]), we narrow our focus to LHE-based MtA for a more efficient solution to CI and robustness.

For robustness. The technique in WMY$^+$23 [59] and WMC24 [58] appears to be a promising choice for robustness. However, it introduces additional rounds or computationally expensive operations during the online phase. 1) The key idea of WMY$^+$23 is to convert the (n', n')-shares of secret nonce k and temporary secret nonce γ into (t, n')-shares, making k_i and γ_i t-threshold. With the help of MtA, $k\gamma$ and $x\gamma$ can also be split into t-threshold shares. As long as t parties contribute honest shares of $k\gamma$ and $x\gamma$, the signature can be successfully generated. However, the t-threshold sharing of k_i and γ_i must be done before running MtA, adding an extra round. Additionally, another round is needed to generate R, making the final protocol five rounds. When factoring in the additional dispute phase induced by CI with private checking, WMY$^+$23 needs 7 rounds, as analyzed in [58]. 2) WMC24 [58] leverages threshold CL encryption to achieve robustness. Their key insight is that threshold CL encryption itself provides the t-threshold access structure needed for robustness. However, this approach requires an extra round for distributively decrypting R, resulting in a four-round scheme, and involves decryption in the online phase, which leads to expensive online computations.

Thus, we begin with LHE-based MtA (specifically, CL-based MtA to avoid range proofs) with public checking and aim to develop a three-round, online-friendly solution for achieving both CI and robustness.

1.2.2. Our Approach. Our solution is based on the *undetermined temporary secret nonce* γ and *simple public checking of* γ.

No addition round with undetermined temporary nonce. Our first observation is that, in WMY$^+$23, the t-threshold sharing of γ in the first round of MtA is the primary reason for adding an extra round before executing the MtAs. Our key insight, however, is that t-threshold sharing of γ is unnecessary for robustness, as it will ultimately be cancelled out during the computation of the s-component. Intuitively, using an undetermined γ (i.e., omitting t-threshold sharing) would eliminate one round of communication. Here, "undetermined" means that the contributors and the final value of γ are not established until the final round, after cheaters have been identified and excluded.

One might be concerned about using an undetermined γ. However, robustness can still be guaranteed if malicious parties are excluded from contributing to γ. Specifically, each party selects a random γ_i and retains it as an additive share of γ whose implicit value is defined as $\sum_{i=1}^{n'} \gamma_i$. If a cheater fails to contribute its share γ_{i^*} honestly, we can identify and remove it, resulting in a new value $\gamma' = \gamma - \gamma_{i^*}$. Fortunately, this does not compromise robustness, as γ' will still be canceled out in the final assembly phase.

Using an undetermined γ allows our protocol to avoid the extra round required by WMY$^+$23 before the MtAs, and by incorporating the new assembly technique from DKLs24, we save an additional round for opening R. Furthermore, public checking of MtA removes the need for two extra rounds during the dispute phase.

Together, these improvements allow our signing protocol to be completed in just three rounds. In the first round, parties broadcast the initial MtA message using additive shares of γ as input while simultaneously generating t-threshold shares of k. In the second round, they use the t-threshold shares of k and x to perform the second round of MtAs. Thus, our offline phase completes within these two rounds, followed by a single online computation round. For further details, please refer to Section 4.

Simple public checking of γ in online phase. One security concern that has not been addressed previously is the consistency of γ. As noted earlier, γ is canceled out in the final assembly phase, which relies on the assembled denominator $k\gamma$ and numerator $m\gamma + rx\gamma$ in the signing equation (1), both using the same value of γ. Therefore, the problem of publicly checking on the consistency of γ should be addressed in the online phase.

The challenge lies in conducting this consistency check efficiently. In previous approaches (e.g., [20], [58], [59]), specific public values like γG or γ's ciphertext were used to verify the consistency of γ. However, in our approach, due to the undetermined nature of γ, there are no predefined public values associated with it.

One possible solution is for each party to prove that the share γ_i concealed within its online message matches the plaintext encrypted during the offline phase. However, this requires costly encryption-based ZKPs, such as the Paillier-based ZKPs used in CGG$^+$20 [20].

Instead, our method directs each party to utilize the EC points $\gamma_i R$ and $\gamma_i(mG + rX)$ to establish a simple Decisional Diffie-Hellman (DDH) relation. This can be achieved with well-established and efficient DDH-ZKPs over the EC

group. Finally, inspired by WMY$^+$23, we use $\gamma_i R$ and $\gamma_i(mG + rX)$ to verify the correctness of each party $\mathcal{P}_i$'s share of $k\gamma$ and $(m\gamma + rx\gamma)$, respectively.

In comparison, CGG$^+$20 requires ZKPs for Paillier encryption, and WMC24 necessitates ZKPs for CL encryption, both of which are significantly more expensive. Although the online consistency check in WMY$^+$23 does not require Paillier or CL operations, it relies on ZKPs that are three times more costly than our DDH-ZKP approach.

1.2.3. Extension to threshold BBS+ signatures. Our method can be extended to thresholding BBS+. Denote by $(\mathbb{G}_1, \mathbb{G}_2, G_1, G_2, q)$ the group-generator-order tuple of the pairing group used in the BBS+ [7] and take $\mathbf{H} = \{H_k\}_{k\in[\ell+1]} \leftarrow\$\ \mathbb{G}_1^{\ell+1}$ as the public parameter. For a message vector $\mathbf{m} = \{m_k\}_{k\in[\ell]} \in \mathbb{Z}_q^\ell$ and the key pair $(x, X = xG_2 \in \mathbb{G}_2)$, the signature is (A, e, s), in which $e, s \in \mathbb{Z}_q$ are public random values and $A \in \mathbb{G}_1$ is computed by $A = \frac{G_1 + s\cdot H_1 + \sum_{k\in[\ell]} m_k \cdot H_{k+1}}{x+e}$. With a temporary secret nonce γ, the above can be written as

$$A = \frac{\gamma(G_1 + s \cdot H_1 + \sum_{k\in[\ell]} m_k \cdot H_{k+1})}{\gamma(x + e)}. \qquad (2)$$

Similar to ECDSA, thresholding BBS+ involves securely computing $\gamma(x+e)$ from the shares of x and γ. In DKL$^+$23 [37], a non-robust threshold BBS+ protocol is developed within three rounds. Recently, WMC24 [58] introduced a robust threshold BBS+ that utilizes threshold CL, though its online phase is comparatively costly due to the threshold decryption protocol. Table 2 summarizes threshold BBS+ constructions.

In the BBS+ signing equation (2), we use the threshold ElGammal technique to generate public nonces e, s, and MtA with public checking to handle the multiplication of $\gamma(x + e)$. Furthermore, our method of threshold ECDSA for CI can be adapted to support threshold BBS+. In this case, checking the consistency of each party's γ_i requires two pairing operations.

1.3. Discussions

Online and overall cost. As illustrated in Fig. 3, our scheme outperforms WMY$^+$23 in both computation and bandwidth, during both the online phase and overall execution. Our online computation is over two orders of magnitude faster than WMC24. However, our online bandwidth is less than WMC24's only when $n \leq 13$. Similarly, our overall cost remains lower than that of WMC24 only in small-scale cases (e.g., $n \leq 12$ for runtime and $n \leq 5$ for bandwidth).

NIST's draft call for threshold schemes categorizes scenarios by the number of parties involved into "two, three, small, medium, large, and enormous"-party profiles [16, Table 3]. Fortunately, small-scale or fewer scenarios are more common in practice. For instance, commercial key custody platforms, such as Coinbase [3], Zengo [5], and Fireblocks [4], typically use a (2,3) threshold, where the signing key is split across three devices: the user's smartphone, laptop, and a server managed by the custody service provider.

Thus, our scheme is highly competitive in two-party, three-party, and small-scale settings, including applications in wallet key custody, decentralized certificate authorities (CAs), decentralized anonymous credentials, and other real-world use cases. Moreover, with our significantly lower online computation costs, our solution is well-suited for resource-constrained environments, such as small embedded systems, mobile devices, and low-power devices.

Attack on WMY$^+$23. As previously mentioned, we identified a vulnerability in WMY$^+$23 that arises when the parties continue the signing process after identifying a cheater: the attacker could potentially obtain the full signing key. This vulnerability prevents WMY$^+$23 from simultaneously achieving robustness and cheater identification, as robustness requires continuing the signing process after cheater identification.

One way to address this vulnerability is to halt the protocol once a cheater is identified, an approach known as identifiable abort [8], [30]. However, this sacrifices robustness. In our solution, we introduce *publicly verifiable* message transmissions, enabling cheater identification without requiring complaints or disputes. This allows the protocol to securely continue and produce the final signature.

Add robustness to CGG$^+$20 & CDKS24. If similar techniques are adopted, CGG$^+$20 [20] can be adjusted to achieve robustness. However, the round count would be at least five, since our undetermined γ cannot be applied on the basis of their four-round framework. Moreover, their online CI process would require much more computational overhead. If one attempts to add robustness to CDKS24 [30], the OT-based MtA with public checking is required. However, it is currently unclear how to construct it. Specifically, the OT-MtA of CDKS24 requires input-revealing (i.e., revealing γ) when identifying cheaters in the MtA calls. If the protocol is not terminated promptly to achieve robustness, this could introduce a vulnerability like [59].

Communication Model. We assume that parties are connected through a synchronous broadcast mechanism, consistent with most prior works. An exception is Groth and Shoup's study [44], which operates under an asynchronous communication model. However, "cheater identification" is inapplicable in an asynchronous context, as it becomes impossible to differentiate between a malicious party that is unresponsive and an honest party experiencing network delays.

Recent Work by Lyu et al. [49]. We have recently taken note of a related paper [49] published on ePrint, which presents a three-round robust threshold ECDSA. Similarly to our work, their work applies the MtA-based paradigm and enables fast online signing. However, it only supports a game-based security proof, whereas ours is accompanied by a simulation-based proof.

1.4. Related Works

Threshold ECDSA. Threshold ECDSA originates from [51] based on Paillier encryption for a two-party setting, then Lindell [47] and Xue et al. [61] improved it. MtA protocol is a core building block of threshold ECDSA for multi-party settings. It falls into two main categories: OT-based [34], [35], [36], and LHE-based [20], [22], [60]. Among these LHE solutions, CL encryption [25] has a unique advantage: it does not require range proofs. However, as analyzed in [55], [57], Paillier and JL need range proofs due to the mismatch between plaintext space (e.g., 3072 bits) and ECDSA's space (e.g., 256 bits). Although Tang et al. [55] employed batching techniques to significantly reduce the overhead of range proofs, the efficiency of Paillier- or JL-based MtA remains inferior to that of the CL-based approach, especially in light of the recent implementation improvements by Bouvier et al. [13]. Therefore, we adopt CL encryption in this work.

For another alternative technical approach, Abram et al. [6] used pseudorandom correlation generators (PCG) [14], [15] to build threshold ECDSA. Their bandwidth complexity is $1 \sim 2$ orders of magnitude smaller than those based on LHE, but their amortized computational cost is expensive.

Boneh et al. [12] presented a universal thresholdizer based on threshold fully homomorphic encryption. It can be used to thresholdize any signature scheme with robustness. Using this tool to construct an efficient threshold ECDSA is worth exploring in the future.

Public Checking. In this work, we consider public checking (aka public verifiability) in order to realize cheater identification without the dispute phase. The concept of public checking is the same as the public verifiability defined in [45]. It ensures that the transmitted messages are publicly verifiable, even if more than threshold (potentially all) parties are corrupt when security or robustness can not be guaranteed. Following this notion, our MtA with public checking can naturally identify cheaters without any complaint or dispute process.

Kate et al. [45] proposed this notion for publicly verifiable secret sharing (PVSS). However, their PVSS is non-interactive, and using it for distributed randomness generation (DRG) can only generate biased randomness, making it unsuitable for threshold ECDSA scenarios. As another independent work, Cascudo and David [21] proposed a PVSS over class groups, which can be used to construct a two-round unbiased DRG protocol. Therefore, this work applies [21]'s PVSS scheme to generate distributed randomness.

1.5. Versions of This Paper

This version differs from the earlier conference version [56] in the following ways.

1. The earlier conference version [56] contained a flaw in the robust threshold BBS+ scheme (Section 5). Specifically, we used a commit-and-release paradigm to generate the public nonces e, s in a distributed manner. This poses no issue in the non-robust case, but in a robust setting, an adversary can bias the distribution of e, s by controlling whether or not some share e_i, s_i is released correctly. In this version, we fix this flaw by leveraging a threshold ElGamal scheme. The threshold decryption guarantees that any t honest parties will jointly produce the values e and s, which are implicitly predetermined and unbiased. We are grateful to Yang Heng for pointing it out.
2. The earlier version adopted Canetti et al. [20]'s functionality of threshold signatures. We have recently become aware of a paper by Cohen et al. [31], which points out that this type of functionality definition suffers from a consistency issue when robustness is added. In this version, we instead adopt an alternative functionality (Figure 2), which has been used in most existing works on threshold ECDSA [35], [36], [48], [60]. We are very grateful to an anonymous reviewer for bringing [31] to our attention.

2. Preliminaries

Let $\mathbb{Z}_q$ be the finite field with q elements $\{0, 1, \cdots, q-1\}$, $[a]$ denote the set $\{1, \cdots, a\}$ for some positive a. By $s \leftarrow\$ S$, we denote that s is chosen uniformly at random from a finite set S. For a number N, $|N|$ denotes the bit length of N; for a set S, $|S|$ denotes the size of S. Let $\lambda \in \mathbb{N}$ and $\lambda_s \in \mathbb{N}$ denote the security parameter and the statistical parameter, respectively

Through this paper, n denotes the number of all participants, and $t \leq n$ denotes the threshold value. For t-threshold shares $\{x_i\}_{i\in[n]}$ of $x \in \mathbb{Z}_q$, we have $x = \sum_{i\in S} \lambda_{i,S} \cdot x_i \bmod q$ for any set $S \subseteq [n]$ with $|S| \geq t$, where $\lambda_{i,S} = \prod_{j\in S\setminus\{i\}} \frac{-j}{i-j}$ is the Lagrange coefficient corresponding to $i \in S$.

2.1. ECDSA

Let $pp = (\mathbb{G}, G, q)$ be the group-generator-order tuple associated with the curve of ECDSA signatures. ECDSA scheme [32] makes use of the hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_q$ and works as follows.

- $\mathsf{KGen}(pp, 1^\lambda) \to (x, X)$: output $x \leftarrow\$ \mathbb{Z}_q$ and $X = xG$.
- $\mathsf{Sign}(x, \mathsf{msg}) \to \sigma$: set $m = \mathcal{H}(\mathsf{msg})$, choose $k \leftarrow\$ \mathbb{Z}_q$ and compute $R = kG$, output $\sigma = (r, s)$ with $r = R|_{x\text{-axis}}$ as the x-projection of point R on $\mathbb{Z}_q$ and $s = k^{-1} \cdot (m + rx) \bmod q$.
- $\mathsf{Verify}(X, (r, s), \mathsf{msg}) \to 0/1$: output 1 if r is the x-projection of $R' = s^{-1}(mG + rX)$ where $m = \mathcal{H}(\mathsf{msg})$.

2.2. Castagnos–Laguillaumie Encryption

Our scheme employs Castagnos–Laguillaumie (CL) encryption [25] as the LHE scheme since it can avoid range proofs required by both Paillier and JL encryptions. CL works in an unknown-order class group $\hat{\mathfrak{G}}$ with order $q\hat{s}$,

only the upper bound $\tilde{s}$ of $\hat{s}$ is given. One can find a subgroup $\mathfrak{F}$ generated by $\mathfrak{f} \in \hat{\mathfrak{G}}$ with the order q. There exists a *deterministic polynomial-time* algorithm $\mathsf{Dlog}(\cdot)$ that returns x on input $\mathfrak{f}^x \in \mathfrak{F}$.

Denote Gen the algorithm that on input the security parameter λ and a prime q, outputs $pp_{\mathsf{CL}} = (\tilde{s}, \mathfrak{f}, \mathfrak{g}_q, \mathfrak{G}, \hat{\mathfrak{G}}, \mathfrak{F})$ such that $\hat{\mathfrak{G}} \supset \mathfrak{G} = \mathfrak{F} \times \mathfrak{G}^q$. $\mathfrak{F} =< \mathfrak{f} >$ is defined as above, $\mathfrak{G}$ is a subgroup of $\hat{\mathfrak{G}}$ generated by $\mathfrak{f}\mathfrak{g}_q$, and $\mathfrak{G}^q =< \mathfrak{g}_q >$ is the subgroup containing all q-th powers in $\mathfrak{G}$.

In CL scheme, we sample the randomness uniformly over the range $\mathcal{D}_q = [0, 2^{\lambda_s} q\tilde{s}]$ whose distribution is at distance less than $2^{-\lambda_s}$ from the uniform distribution in $\mathbb{Z}_{q\tilde{s}}$, where λ_s is the statistical parameter. For security to hold, the secret key is sampled uniformly over $\mathcal{D}$, s.t. $\{(\mathfrak{f}\mathfrak{g}_q)^r, r \leftarrow\!\$\ \mathcal{D}\}$ is at a distance less than $2^{-\lambda}$ from the uniform distribution in $\mathfrak{G}$ [23]. The CL scheme and linear homomorphic operations are described below.

- $\mathsf{KGen}(1^\lambda) \to (ek, dk)$: output $dk \leftarrow\!\$\ \mathcal{D}$ and $ek = \mathfrak{g}_q^{dk}$.
- $\mathsf{Enc}(ek, m; \rho) \to C_m$: pick $\rho \leftarrow\!\$\ \mathcal{D}_q$ and output $C_m = (\mathfrak{g}_q^\rho, \mathfrak{f}^m ek^\rho)$.
- $\mathsf{Dec}(dk, (c_1, c_2)) \to m$: output $\mathsf{Dlog}(c_2/c_1^{dk})$.
- Addition: $C_m \oplus C_{m'} \to C_{m+m'}$: parse $C_m = (c_1, c_2)$ and $C_{m'} = (c_1', c_2')$, output $C_{m+m'} = (c_1 \cdot c_1', c_2 \cdot c_2')$.
- Constant Multiplication: $a \odot C_m \to C_{am}$: parse $C_m = (c_1, c_2)$, output $C_{am} = (c_1^a, c_2^a)$.

2.3. Zero-Knowledge Proof

Let x be a statement of an NP language $\mathcal{L}$ and w be its witness. A zero-knowledge proof (ZKP) allows a prover to convince the verifier that $(x, w) \in \mathcal{R}$ without revealing w. Σ-protocol $(\mathsf{Prove}, \mathsf{Verify})$ is a special ZKP protocol and can be transferred to non-interactive via Fiat–Shamir heuristic [38]. Refer to Appendix A.1 for details.

- $\mathsf{Prove}_{\mathcal{R}}(x; w) \to \pi$: On input of statement x and the secret witness w, the probabilistic algorithm returns a proof π.
- $\mathsf{Verify}_{\mathcal{R}}(x; \pi) \to 0/1$: On input statement x and proof π, the deterministic algorithm returns a bit, indicating the validity of π.

Our scheme needs ZKPs for the following NP-relations. Their instantiations are Σ-protocols, given in Appendix A.2.

CL Key Pair. $\mathcal{R}_{\mathsf{key}} = \{(ek, dk) : dk \in \mathcal{D}, ek = \mathfrak{g}_q^{dk}\}$.

Well-formedness of a CL Ciphertext. Given ek as CL's encryption key, define

$$\mathcal{R}_{\mathsf{Enc}} = \{(C_m, (m \in \mathbb{Z}_q, \rho \in \mathcal{D}_q)) : C_m = \mathsf{Enc}(ek, m; \rho)\}.$$

Validity of Encrypted Shares. Given $\{ek_j\}_{j \in S}$ as a set of CL's encryption keys and threshold t, define[4]

$$\mathcal{R}_{\mathsf{Sh}} = \{(\{C_{f(j)}\}_{j \in S}, (f(\cdot), \rho)) : \mathsf{deg}(f(\cdot)) \le t-1, C_{f(j)} = \mathsf{Enc}(ek_j, f(j); \rho)\}.$$

CL Decryption and Discrete Logarithm.

$$\mathcal{R}_{\mathsf{Dec\text{-}DL}} = \{((C_a, A), (a \in \mathbb{Z}_q, dk \in \mathcal{D})) : A = aG, a = \mathsf{Dec}(dk, C_a)\}.$$

4. $\mathsf{deg}(f(\cdot))$ is the degree of polynomial $f(\cdot)$.

- Round 1: Each party $\mathcal{P}_i$ broadcasts the *shares distribution* message $(\{C_{x_{j,i}}\}_{j \in S}, \pi_i) \leftarrow \mathsf{ShareDist}(\{ek_j\}_{j \in S}, t)$:
  a) pick a $(t-1)$-degree polynomial $f(\cdot)$ and $\rho \leftarrow\!\$\ \mathcal{D}_q$
  b) compute $C_{x_{j,i}} = \mathsf{Enc}(ek_j, f(j); \rho)$ for each $j \in S$
  c) generate $\pi_i \leftarrow \mathsf{Prove}_{\mathsf{Sh}}(\{C_{x_{j,i}}\}_{j \in S}, (f(\cdot), \rho))^a$.
- Round 2: After receiving $(\{C_{x_{\nu,j}}\}_{\nu \in S}, \pi_j)$ from each $\mathcal{P}_j$, $\mathcal{P}_i$ first verifies π_j and sets $S = S \setminus \{j\}$ if it fails. $\mathcal{P}_i$ generates the *shares combination* message $(x_i, X_i, \pi_i') \leftarrow \mathsf{ShareComb}(\{C_{x_{i,j}}\}_{j \in S}, dk_i)$:
  a) decrypt $x_{i,j} \leftarrow \mathsf{Dec}(dk_i, C_{x_{i,j}})$ for each $j \in S$
  b) compute $x_i = \sum_{j \in S} x_{i,j} \mod q$ and $X_i = x_i G$.
  c) generate $\pi_i' \leftarrow \mathsf{Prove}_{\mathsf{Dec\text{-}DL}}((C_{x_i}, X_i), (x_i, dk_i))$ with $C_{x_i} = \oplus_{j \in S} C_{x_{i,j}}$.
  Broadcast (X_i, π_i').
- Output: On receiving (X_j, π_j') from each $\mathcal{P}_j$, verify π_j' and set $S = S \setminus \{j\}$ if it fails. $\mathcal{P}_i$ stores its own secret share x_i and others' public shares $\{X_j\}_{j \in S}$, as well as $X = \sum_{j \in S} \lambda_{j,S} X_j$.

a. The $\{C_{x_{j,i}}\}_{j \in S}$'s validity proof π_i is constant-size, proving that their plaintexts come from a same polynomial of degree $\le t-1$ [21].

Figure 1: Distributed randomness generation protocol

DDH Relation.

$$\mathcal{R}_{\mathsf{DDH}} = \{((G, A, B, C) \in \mathbb{G}^4, a \in \mathbb{Z}_q) : A = aG, C = aB\}.$$

2.4. Distributed Randomness Generation

Distributed randomness generation (DRG) enables each participant to obtain a t-threshold share x_i for a secret value $x \leftarrow\!\$\ \mathbb{Z}_q$, along with the corresponding EC point $X = xG$ and each public share $X_i = x_i G$ being revealed. In this work, we invoke [21]'s publicly verifiable secret sharing (PVSS) over the class group to implement DRG, where public verification allows us to expose cheaters openly.

The participants are $\{\mathcal{P}_j\}_{j \in S}$, where S is a set of indices. The CL encryption is employed, with ek_j being each party $\mathcal{P}_j$'s public key generated via correctly running KGen[5]. Given a set of CL encryption keys $\{ek_j\}_{j \in S}$ and a threshold value t, the DRG is a two-round protocol as illustrated in Figure 1.

3. Multiplicative-to-Additive (MtA) Protocol

3.1. Private vs Public Checking of MtA

MtA takes inputs a and b from two parties, such as Alice and Bob, and securely computes α for Alice and β for Bob such that $\alpha + \beta = ab \pmod q$. Simultaneously, a consistency-checking mechanism is carried out alongside the MtA computation, where Bob provides additional proof to convince Alice (or all parties) that his input b is correct

5. Public Key Infrastructure (PKI) setup is established by making each party produce a $\mathcal{R}_{\mathsf{key}}$-relation proof for ek_j in the key generation phase of threshold ECDSA.

(as his secret key share). Using CL encryption, the MtA functionality can be implemented as follows.

- Alice sends a's ciphertext C_a and its well-formedness proof to Bob;
- Bob picks a random β, generates $-\beta$'s ciphertext $C_{-\beta}$ under Alice's public key, and replies with the ciphertext $C_\alpha = b \odot C_a \oplus C_{-\beta}$;
- Alice decrypts C_α to obtain $\alpha = b \cdot a - \beta \bmod q$.

Assuming that Bob's input b is the discrete logarithm of a public point $X_{\text{Bob}} = bG$, the consistency checking mechanism falls into the following two methods:

- Private checking (as done in [24], [59]): Bob only needs to convince Alice (also known as pairwise checking as described in [36]). In the second round, Bob sends C_α along with $B_{\text{Bob}} = \beta G$. After this, only Alice can check $\alpha G + B_{\text{Bob}} \stackrel{?}{=} aX_{\text{Bob}}$ with her private inputs α, a.
- Public checking (as done in [20], [41]): Bob aims to convince all parties by sending the ciphertext C_α along with a ZKP that b corresponds to the discrete logarithm of X_{Bob}.

MtA with public checking can easily facilitate cheater identification (CI), as anyone can openly verify Bob's actions. Conversely, private checking achieves this by revealing Alice's private information.

3.2. Vulnerability of WMY$^+$23 [59]

We demonstrate the vulnerability of WMY$^+$23 [59] when private checking is used to achieve both CI and robustness. In WMY$^+$23, Alice publicly discloses α along with a proof that α corresponds to the decryption of C_α, and the EC point $A = aX_{\text{Bob}}$ along with its correctness proof. Then anyone can find Bob's misbehavior by checking $\alpha G + B_{\text{Bob}} \neq A$.

We demonstrate that their approach introduces a security vulnerability: an adversary $\mathcal{A}$ could obtain the signing key x by controlling $t-1$ parties, which is allowed under a (t, n) threshold signature scheme.

1. Assume $\mathcal{A}$ corrupts $t-1$ parties and one of them is Bob. $\mathcal{A}$ executes all the corrupted parties honestly, but Bob.
2. $\mathcal{A}$ playing the role of Bob misbehaves during one MtA process, causing an honest party $\mathcal{P}_{i^*}$ (as Alice) to reveal her private value α for CI.
3. $\mathcal{A}$ can compute $\mathcal{P}_{i^*}$'s private input a of MtA as $b^{-1}(\alpha + \beta) \bmod q$ where b is Bob's input and β is the output received from MtA. In WMY$^+$23, a is $\mathcal{P}_{i^*}$'s share of the secret nonce k, denoted by k_{i^*}.
4. $\mathcal{A}$ can recover the secret nonce k by piecing together the $t-1$ shares controlled by itself and k_{i^*}.
5. When robustness is required, the protocol processes if the number of honestly performing signatories is no less than t. Finally, a valid ECDSA signature (r, s) is generated. With the knowledge of k, $\mathcal{A}$ can recover the signing key $x = r^{-1}(ks - \mathcal{H}(\mathsf{msg})) \bmod q$ accordingly.

We remark that the above attack does not affect [6], [8], [24], [30], [36]. Since they do not consider robustness, thus the protocols do not continue if a cheater exists.

3.3. Multi-Party MtA with Public Checking

As shown above, MtA with public checking requires a ZKP to prove the correctness of Bob's ciphertext C_α. Here, we use a batching technique to aggregate ZKPs in the multi-party case.

Multi-Party MtA. For $n > 2$ signatories, threshold ECDSA requires the computation of $(\sum_{i\in S} \lambda_i k_i) \cdot (\sum_{i\in S} \lambda_i \gamma_i)$. Each party holding k_i, γ_i performs MtA with each other holding k_j, γ_j. Threshold ECDSA [20], [41] invokes MtAs in the following manner:

- Each party $\mathcal{P}_i$ broadcasts γ_i's ciphertext C_{γ_i} and its well-formedness proof.
- After receiving C_{γ_j} from each $\mathcal{P}_j$, $\mathcal{P}_i$ picks $\beta_{i,j} \leftarrow\!\!\$\ \mathbb{Z}_q$ and generates $C_{-\beta_{i,j}}$ under $\mathcal{P}_j$'s public key. Broadcast $C_{\alpha_{j,i}} \leftarrow k_i \odot C_{\gamma_j} \oplus C_{-\beta_{i,j}}$ and its correctness proof $\tau_{i,j}$[6].

In the multi-party MtA protocol, $\mathcal{P}_i$ performs homomorphic operations on each $\mathcal{P}_j$'s C_{γ_j} using the same k_i and generates $\tau_{i,j}$ for each $j \in S$.

Here, we extend the batching technique for Paillier in [55] to the proofs of CL ciphertexts, *aggregating* $\{\tau_{i,j}\}_{j\in S}$ into a single one with *batch-proving* and *batch-verification*.

With the batching technique, our multi-party MtA protocol with public checking is illustrated in the following, which can be denoted as $\mathsf{MPMtA}(\mathcal{P}_i(k_i, \gamma_i)) \to \{\delta_{i,j}\}_{j\in S}$ from $\mathcal{P}_i$'s view.

- Round 1: Each party $\mathcal{P}_i$ broadcasts $(C_{\gamma_i}, \tau_i) \leftarrow \mathsf{MPMtA}_1(\gamma_i, ek_i)$:
  a) pick $\rho \leftarrow\!\!\$\ \mathcal{D}_q$, generate $C_{\gamma_i} \leftarrow \mathsf{Enc}(ek_i, \gamma_i; \rho)$
  b) generate $\tau_i \leftarrow \mathsf{Prove}_{\mathsf{Enc}}(C_{\gamma_i}, (\gamma_i, \rho))$.
- Round 2: After receiving (C_{γ_j}, τ_j) from each $\mathcal{P}_j$, $\mathcal{P}_i$ first verifies τ_j and sets $S = S \setminus \{j\}$ if it fails. $\mathcal{P}_i$ runs $\mathsf{MPMtA}_2(\{C_{\gamma_j}\}_{j\in S}, k_i) \to (\{C_{\alpha_{j,i}}, \beta_{i,j}\}_{j\in S}, \tau_i')$:
  a) pick $e \leftarrow\!\!\$\ \mathcal{D}_q$, let $k_i^* = k_i + eq$ and $R_i = k_i G$
  b) for each $j \in S$, parse $C_{\gamma_j} = (c_{j,1}, c_{j,2})$, pick $\beta_{i,j} \leftarrow\!\!\$\ \mathbb{Z}_q$, and compute $C_{\alpha_{j,i}} = (c_{j,1}^{k_i^*}, c_{j,2}^{k_i^*} \mathfrak{f}^{-\beta_{i,j}})$, $B_{i,j} = \beta_{i,j} G$
  c) $\mathsf{Prove}_{\mathsf{m\text{-}AffDL}}((\{C_{\gamma_j}, C_{\alpha_{j,i}}, B_{i,j}\}_{j\in S}, R_i), (\{\beta_{i,j}\}_{j\in S}, k_i^*))$ is invoked to generate τ_i' (defined below).

  Broadcast $(\{C_{\alpha_{j,i}}, B_{i,j}\}_{j\in S}, R_i, \tau_i')$.
- Output: After receiving τ_j' from each $\mathcal{P}_j$, verify it and set $S = S \setminus \{j\}$ if it fails. $\mathcal{P}_i$ gets output $\{\delta_{i,j}\}_{j\in S} \leftarrow \mathsf{MPMtAOut}(dk_i, \{\beta_{i,j}, C_{\alpha_{i,j}}\}_{j\in S})$:
  a) decrypt $\alpha_{i,j} = \mathsf{Dec}(dk_i, C_{\alpha_{i,j}})$ for each $j \in S$
  b) compute $\delta_{i,j} = \alpha_{i,j} + \beta_{i,j} \bmod q$ for each $j \in S$.

In our multi-party MtA, the privacy of γ_i is ensured by the CL ciphertext C_{γ_i}; the privacy of k_i is preserved through the security of the affine operations, as analyzed further below.

Security of affine operations. Our CL affine operation enables us to generate a constant-size proof for multiple affine ciphertexts and can be defined as an algorithm $\mathsf{Aff}(\{C_{\gamma_j} = (c_{j,1}, c_{j,2})\}_{j\in S}, k \in \mathbb{Z}_q) \to \{C_{\alpha_j}\}_{j\in S}$:

- pick $e \leftarrow\!\!\$\ \mathcal{D}_q$, let $k^* = k + eq$

6. All honest parties need to verify this proof for identifying cheaters.

- output $\{C_{\alpha_j} = (c_{j,1}^{k^*}, c_{j,2}^{k^*} \mathfrak{f}^{-\beta_j})\}_{j\in S}$ with each $\beta_j \leftarrow\$ \mathbb{Z}_q$.

Definition 1. *The affine algorithm* Aff *is secure, if for any PPT adversary $\mathcal{A}$ who controls $t-1$ parties $\{\mathcal{P}_j\}_{j\in S_c}$, its winning probability is at most* $\mathsf{negl}(\lambda)$ *away from 1/2 in the following experiment.*

1. *Generate honest parties' $\{C_{\gamma_j}\}_{j\in S\setminus S_c}$ and receive (well-formed) $\{C_{\gamma_j}\}_{j\in S_c}$ from $\mathcal{A}$.*
2. *After receiving $(k^{(0)}, k^{(1)})$ from $\mathcal{A}$, pick $b \leftarrow\$ \{0,1\}$, generate $\{C_{\alpha_j}\}_{j\in S} \leftarrow \mathsf{Aff}(\{C_{\gamma_j}\}_{j\in S}, k^{(b)})$.*
3. *$b' \leftarrow \mathcal{A}(\{C_{\alpha_j}, C_{\gamma_j}\}_{j\in S})$, $\mathcal{A}$ wins if $b' == b$.*

Intuitively, the security of our affine algorithm can be derived from the security of CL encryption, as shown in the following theorem.

Theorem 1. *Based on the hard subgroup membership (HSM) assumption (Definition 6), our affine algorithm* Aff *in multi-party MtA is secure according to Definition 1.*

Refer to Appendix B for the proof.

Proof of correct affine operations. Define $\mathcal{R}_{\mathsf{m\text{-}AffDL}}$

$$\mathcal{R}_{\mathsf{m\text{-}AffDL}} = \{((\{C_{\gamma_j}, C_{\alpha_j}, B_j\}_{j\in S}, R), (\{\beta_j \in \mathbb{Z}_q\}_{j\in S}, k^*)) : R = (k^* \bmod q)G, \{B_j = \beta_j G, C_{\alpha_j} = (c_{j,1}^{k^*}, c_{j,2}^{k^*}\mathfrak{f}^{-\beta_j})\}_{j\in S}\}.$$

Given random $e_j \leftarrow\$ [0, 2^\lambda]$ for each $j \in S$, the prover and verifier can call $\mathsf{Aggregate}(\{e_j, C_{\gamma_j}, C_{\alpha_j}, B_j\}_{j\in S}) \rightarrow (C_{\gamma^*} = (c_1, c_2), C_{\alpha^*} = (d_1, d_2), B)$ as below

- $\{C_{\gamma_j} = (c_{j,1}, c_{j,2})\}_{j\in S}$: $c_1 = \prod_{j\in S} c_{j,1}^{e_j}$, $c_2 = \prod_{j\in S} c_{j,2}^{e_j}$
- $\{C_{\alpha_j} = (d_{j,1}, d_{j,2})\}_{j\in S}$: $d_1 = \prod_{j\in S} d_{j,1}^{e_j}$, $d_2 = \prod_{j\in S} d_{j,2}^{e_j}$
- $\{B_j\}_{j\in S}$: $B = \sum_{j\in S} e_j B_j$

With $\beta = \sum_{j\in S} e_j\beta_j \bmod q$, $\mathcal{R}_{\mathsf{m\text{-}AffDL}}$ is compressed into

$$\mathcal{R}^* = \{((C_{\gamma^*}, C_{\alpha^*}, B, R), (\beta, k^*)) : B = \beta G, R = (k^* \bmod q)G, d_1 = (c_1)^{k^*}, d_2 = (c_2)^{k^*}\mathfrak{f}^{-\beta})\}.$$

Please refer to Appendix A.3 for the instantiation, whose size is constant.

4. Robust Threshold ECDSA Scheme

4.1. Distributed Key Generation

The key generation phase establishes the PKI setup and invokes the distributed randomness generation (DRG) protocol (c.f. Section 2.4). Let $\hat{S} = [n]$ be the set of indices of all parties. It could dynamically decrease with each round, as DRG can identify the cheaters publicly.

- Round 1: $\mathcal{P}_i$ runs CL's key generation $\mathsf{KGen}(1^\lambda) \rightarrow (ek_i, dk_i)$ and generates the proof $\bar{\pi}_i \leftarrow \mathsf{Prove}_{\mathsf{key}}(ek_i, dk_i)$. Broadcast $ek_i, \bar{\pi}_i$.
- Round 2: Upon receiving $\bar{\pi}_j$ from each $\mathcal{P}_j$, $\mathcal{P}_i$ sets $\hat{S} = \hat{S} \setminus \{j\}$ if the verification fails. $\mathcal{P}_i$ broadcasts the shares-distribution message $(\{C_{x_{j,i}}\}_{j\in\hat{S}}, \pi_i) \leftarrow \mathsf{ShareDist}(\{ek_j\}_{j\in\hat{S}}, t)$.
- Round 3: On receiving π_j from each $j \in \hat{S}$, set $\hat{S} = \hat{S} \setminus \{j\}$ if the verification fails. $\mathcal{P}_i$ does:
  a) invoke $\mathsf{ShareComb}(\{C_{x_{i,j}}\}_{j\in\hat{S}}, dk_i) \rightarrow (x_i, X_i, \pi_i')$
  b) broadcast (X_i, π_i').
- Key Generation Output: Upon receiving $\{X_j, \pi_j'\}_{j\in\hat{S}}$, $\mathcal{P}_i$ sets $\hat{S} = \hat{S} \setminus \{j\}$ if π_j' is not valid. Each party gets the output:
  a) secret key share x_i, others' public key shares $\{X_j\}_{j\in\hat{S}}$
  b) ECDSA public key $X = \sum_{j\in\hat{S}} \lambda_{j,\hat{S}} X_j$.

Notations	Descriptions
x; x_i	signing key; $\mathcal{P}_i$'s t-threshold share of x
k; k_i	random nonce; $\mathcal{P}_i$'s t-threshold share of k
γ_i	an additive share of γ picked by $\mathcal{P}_i$
$\{\delta_{i,j}\}_{j\in S}$	$\mathcal{P}_i$'s share of $k\gamma$, generated via $\mathsf{MPMtA}(\mathcal{P}_i(k_i, \gamma_i))$
$\{\zeta_{i,j}\}_{j\in S}$	$\mathcal{P}_i$'s share of $x\gamma$, generated via $\mathsf{MPMtA}(\mathcal{P}_i(x_i, \gamma_i))$
$\{\chi_{i,j}\}_{j\in S}$	$\mathcal{P}_i$'s share of $(m+rx)\gamma$, $\chi_{i,j} = m\gamma_i + r\zeta_{i,j}$
$\hat{S}$; S ($S \subseteq \hat{S}$)	indices set of key generation parties; signing parties
X; R	public key of ECDSA; pre-signature $R = kG$
C_{**}	ciphertext of $**$
$B_{i,j}$; $\hat{B}_{i,j}$	$B_{i,j} = \beta_{i,j}G$; $\hat{B}_{i,j} = \hat{\beta}_{i,j}G$
D_i; Γ_i	$D_i = \gamma_i R$; $\Gamma_i = \gamma_i(mG + rX)$

TABLE 3: Some notations and corresponding descriptions.

4.2. Distributed Signing

4.2.1. Intuitions. We present our protocol using notions listed in Table 3. In the offline phase, the parties can invoke DRG (c.f. Figure 1) to give each one a t-threshold share of k_i and public $R = kG$. Also, the parties need to compute

$$(\sum_{i\in S} \lambda_{i,S} k_i) \cdot (\sum_{i\in S} \lambda_{i,S}\gamma_i) \text{ and } (\sum_{i\in S} \lambda_{i,S} x_i) \cdot (\sum_{i\in S} \lambda_{i,S}\gamma_i).$$

To achieve robustness, we first neglect the Lagrange coefficients when invoking MtA. For each $\mathcal{P}_i$, it gets the outputs $\{\delta_{i,j}\}_{j\in S} \leftarrow \mathsf{MPMtA}(\mathcal{P}_i(k_i, \gamma_i))$, $\{\zeta_{i,j}\}_{j\in S} \leftarrow \mathsf{MPMtA}(\mathcal{P}_i(x_i, \gamma_i))$ (c.f., Section 3.3).

In the above two MtA invocations, γ_i is a common input, so its ciphertext can serve as the first-round MtA message and be used in both of the two MtA processes. Then, k_i is the second-round input for MtA, at which point each party has already obtained t-threshold k_i from DRG. Therefore, DRG and MtA are executed concurrently and completed in two rounds.

After knowing the message $m \in \mathbb{Z}_q$, each party broadcasts its shares of $k\gamma$ and $(m + xr)\gamma$ after masking. Only in the final assembly phase is the set of honest parties determined, and it is at this point that we involve the Lagrange coefficients. At this point, the value of γ becomes determined. It will be canceled as long as consistency is maintained in the numerator and denominator. To ensure consistency, each $\mathcal{P}_i$ must prove that its shares hide the same γ_i. This work employs a simple and cheap DDH-ZKP to ensure it.

4.2.2. Concrete Constructions. Here, we present the specifications of our signing protocol.

Offline Phase.

- Round 1: $\mathcal{P}_i$ does the following

a) $(C_{\gamma_i}, \tau_i) \leftarrow \mathsf{MPMtA}_1(\gamma_i, ek_i)$ with $\gamma_i \leftarrow\$ \mathbb{Z}_q$
b) generate $(\{C_{k_{j,i}}\}_{j\in S}, \pi_i) \leftarrow \mathsf{ShareDist}(\{ek_j\}_{j\in S}, t)$
c) broadcast $(C_{\gamma_i}, \tau_i, \{C_{k_{j,i}}\}_{j\in S}, \pi_i)$.

- Round 2: On receiving τ_j, π_j from each $\mathcal{P}_j$, verify them and set $S = S \setminus \{j\}$ if one of them fails. $\mathcal{P}_i$ does
a) $(k_i, R_i, \pi'_i) \leftarrow \mathsf{ShareComb}(\{C_{k_{i,j}}\}_{j\in S}, dk_i)$
b) $(\{C_{\alpha_{j,i}}, \beta_{i,j}\}_{j\in S}, \tau'_i) \leftarrow \mathsf{MPMtA}_2(\{C_{\gamma_j}\}_{j\in S}, k_i)$
c) $(\{C_{\hat{\alpha}_{j,i}}, \hat{\beta}_{i,j}\}_{j\in S}, \hat{\tau}'_i) \leftarrow \mathsf{MPMtA}_2(\{C_{\gamma_j}\}_{j\in S}, x_i)$
d) $B_{i,j} = \beta_{i,j}G, \hat{B}_{i,j} = \hat{\beta}_{i,j}G$ for each $j \in S$
e) broadcast $(\{C_{\alpha_{j,i}}, C_{\hat{\alpha}_{j,i}}, B_{i,j}, \hat{B}_{i,j}\}_{j\in S}, R_i, \pi'_i, \tau'_i, \hat{\tau}'_i)$.

- Offline Output: Upon receiving proofs $\pi'_j, \tau'_j, \hat{\tau}'_j$ from each $\mathcal{P}_j$, set $S = S \setminus \{j\}$ if one of them is invalid. $\mathcal{P}_i$ gets the output
a) $\{\delta_{i,j}\}_{j\in S} \leftarrow \mathsf{MPMtAOut}(dk_i, \{\beta_{i,j}, C_{\alpha_{i,j}}\}_{j\in S})$
b) $\{\zeta_{i,j}\}_{j\in S} \leftarrow \mathsf{MPMtAOut}(dk_i, \{\hat{\beta}_{i,j}, C_{\hat{\alpha}_{i,j}}\}_{j\in S})$
c) $R = \sum_{j\in S} \lambda_{j,S} R_j$, $r = R|_{x\text{-axis}} \bmod q$.

Online Phase. On input the message to be signed $\mathsf{msg} \in \{0,1\}^*$, set $m = \mathcal{H}(\mathsf{msg}) \in \mathbb{Z}_q$. In the online phase, each $\mathcal{P}_i$ will use its own secret shares $(k_i, x_i, \gamma_i, \{\delta_{i,j}, \zeta_{i,j}\}_{j\in S})$ and public shares $\{B_{j,\nu}, \hat{B}_{j,\nu}\}_{j,\nu\in S}$[7] received in Round 2.

- One-Round Interaction: $\mathcal{P}_i$ does
a) pick two random $(t-1)$-degree polynomials $f(\cdot), f'(\cdot)$ with $f(0) = f'(0) = 0$
b) mask $\delta_{i,j}$: $\delta_{i,j} = \delta_{i,j} + f(j) \bmod q$ for each $j \in S$
c) generate $(m + rx)\gamma$'s share: $\{\chi_{i,j} = m\gamma_i + r\zeta_{i,j} + f'(j) \bmod q\}_{j\in S}$
d) generate $D_i = \gamma_i R, \Gamma_i = \gamma_i(mG + rX)$
e) generate $\psi_i \leftarrow \mathsf{Prove}_{\mathsf{DDH}}((R, D_i, mG + rX, \Gamma_i), \gamma_i)$
f) broadcast $(\{\delta_{i,j}, \chi_{i,j}\}_{j\in S}, \psi_i)$.

- Online Output: On receiving $\{\delta_{j,\nu}, \chi_{j,\nu}\}_{\nu\in S}$ for each $j \in S$, assemble $s = \dfrac{\sum_{j,\nu\in S} \lambda_{j,S}\lambda_{\nu,S} \cdot \chi_{j,\nu}}{\sum_{j,\nu\in S} \lambda_{j,S}\lambda_{\nu,S} \cdot \delta_{j,\nu}}$ if $|S| \geq t$. Run $b \leftarrow \mathsf{Verify}(X, (r, s), \mathsf{msg})$. It distinguishes two cases.
a) If $b = 1$, output the signature (r, s).
b) Else, the honest parties identify the cheaters:
i) generate $D_j = \sum_{\nu\in S} \lambda_{\nu,S}(\delta_{j,\nu}G - B_{j,\nu} + B_{\nu,j})$
ii) generate $\Gamma_j = \sum_{\nu\in S} \lambda_{\nu,S}(\chi_{j,\nu}G - r\hat{B}_{j,\nu} + r\hat{B}_{\nu,j})$
iii) verify the DDH-ZKP ψ_j for the statement $(R, D_j, mG + rX, \Gamma_j)$ and set $S = S \setminus \{j\}$ if it fails.

If $|S| \geq t$ still holds, re-assemble $s = \dfrac{\sum_{j,\nu\in S} \lambda_{j,S}\lambda_{\nu,S} \cdot \chi_{j,\nu}}{\sum_{j,\nu\in S} \lambda_{j,S}\lambda_{\nu,S} \cdot \delta_{j,\nu}}$ and output (r, s).

Case b) is required only when misbehavior exists.

4.3. Analysis of CI & Robustness

Cheater Identification. The CI property is obviously guaranteed in the key generation and offline signing phases since we invoke *publicly verifiable* secret sharing and MtA with *public checking*. For the online phase, we give more details.

The proof ψ_j ensures that D_j, Γ_j are generated with the same γ_j. The honestly generated $\{\delta_{j,\nu}\}_{\nu\in S}$ should satisfy $\sum_{\nu\in S} \lambda_{\nu,S}(\delta_{j,\nu} - \beta_{j,\nu} + \beta_{\nu,j}) = \sum_{\nu\in S} \lambda_{\nu,S}(\gamma_j k_\nu) = \gamma_j k$. And the honestly generated $\{\chi_{j,\nu}\}_{\nu\in S}$ should satisfy $\sum_{\nu\in S} \lambda_{\nu,S}(\chi_{j,\nu} - r\hat{\beta}_{j,\nu} + r\hat{\beta}_{\nu,j}) = \sum_{\nu\in S} \lambda_{\nu,S}(m\gamma_j + r\gamma_j x_\nu) = m\gamma_j + rx\gamma_j$. Thus, if some party $\mathcal{P}_j$ contributes incorrect shares $\{\delta_{j,\nu}, \chi_{j,\nu}\}_{\nu\in S}$ in the online phase, it will be identified by our CI mechanism.

Robustness. In the offline phase, each party uses x_i, k_i, γ_i to call MtAs without considering Lagrange coefficient. If one malicious party $\mathcal{P}_j$ is detected, the honest party $\mathcal{P}_i$ can freely discard the shares $\delta_{i,j}, \zeta_{i,j}$ generated in collaboration with $\mathcal{P}_j$. As long as $|S| \geq t$, the offline phase must output $R = \sum_{j\in S} \lambda_{j,S} R_j = kG$.

The online phase can assemble $\sum_{j,\nu\in S} \lambda_{j,S}\lambda_{\nu,S}\delta_{j,\nu} = \sum_{j,\nu\in S} \lambda_{j,S}\lambda_{\nu,S}\gamma_j k_\nu = (\sum_{j\in S} \lambda_{j,S}\gamma_j)(\sum_{\nu\in S} \lambda_{\nu,S}k_\nu)$. It also assembles $\sum_{j,\nu\in S} \lambda_{j,S}\lambda_{\nu,S}\chi_{j,\nu} = m\sum_{j\in S} \lambda_{j,S}\gamma_j + r(\sum_{j\in S} \lambda_{j,S}\gamma_j)(\sum_{\nu\in S} \lambda_{\nu,S}x_\nu)$. If $|S| \geq t$ still holds, we get $s = \dfrac{m + rx}{k}$ with $\sum_{j\in S} \lambda_{j,S}\gamma_j$ being canceled.

5. Robust Threshold BBS+ Scheme

Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ be a paring group with the generators $G_1 \in \mathbb{G}_1, G_2 \in \mathbb{G}_2$ and an efficient pairing operation $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, such that for any $x, y \in \mathbb{Z}_q$, $\mathsf{e}(x \cdot G_1, y \cdot G_2) = xy \cdot \mathsf{e}(G_1, G_2)$. Let pp be public parameters containing $(\mathbb{G}_1, \mathbb{G}_2, G_1, G_2, q)$ and $\ell + 1$ random group elements $\mathbf{H} = \{H_k\}_{k\in[\ell+1]} \leftarrow\$ \mathbb{G}_1^{\ell+1}$. BBS+ signing a vector of ℓ messages works as follows.

- $\mathsf{KGen}(pp, 1^\lambda) \to (x, X)$: $x \leftarrow\$ \mathbb{Z}_q$, $X = xG_2 \in \mathbb{G}_2$.
- $\mathsf{Sign}(x, \mathbf{m} = \{m_k\}_{k\in[\ell]}) \to \sigma$: pick $e, s \leftarrow\$ \mathbb{Z}_q$, and compute $A = \frac{G_1 + sH_1 + \sum_{k\in[\ell]} m_k H_{k+1}}{x+e} \in \mathbb{G}_1$, and output $\sigma = (A, e, s)$.
- $\mathsf{Verify}(X, \sigma = (A, e, s), \mathbf{m}) \to 0/1$: output 1 if $\mathsf{e}(A, X + eG_2) = \mathsf{e}(G_1 + sH_1 + \sum_{k\in[\ell]} m_k H_{k+1}, G_2)$.

The key generation phase of threshold BBS+ is the same as that of threshold ECDSA in Section 4.1.

For the signing protocol, we will use a threshold ElGamal (in the exponent) encryption scheme over the EC group $(\mathbb{G}, G, q)$. The syntax is as follows.

- Distributed key generation: The public key is an EC group element $\hat{X} \in \mathbb{G}$. Each party $\mathcal{P}_i$ holds a t-threshold share $\hat{x}_i$ of $\hat{X}$'s discrete logarithm $\hat{x} \in \mathbb{Z}_q$, along with each public share $\hat{X}_i = \hat{x}_i G$ being revealed. The distributed generation can be realized with the DRG protocol, invoked in the DKG phase of threshold BBS+.
- $\mathsf{El.Enc}(M, \hat{X}) \to (B_1, B_2)$: Given the plaintext $M \in \mathbb{G}$ and public key $\hat{X} \in \mathbb{G}$, pick a randomness $r \leftarrow\$ \mathbb{Z}_q$, compute the ciphertext $B_1 = rG, B_2 = M + r\hat{X}$.
- $\mathsf{El.PartDec}((B_1, B_2), \hat{x}_i) \to M_i$: Given the ciphertext (B_1, B_2) and secret key share $\hat{x}_i$, the party $\mathcal{P}_i$ can compute the partial decryption result $M_i = B_2 - \hat{x}_i B_1$.
- $\mathsf{El.Combine}(\{M_i\}_{i\in S}) \to M$: if $|S| \geq t$, compute $M = \sum_{i\in S} \lambda_{i,S} M_i$.

Offline Phase.

- Round 1: $\mathcal{P}_i$ does the following

7. $\{B_{j,\nu}, \hat{B}_{j,\nu}\}_{\nu\in S}$'s correctness is guaranteed by the proofs $\tau'_j, \hat{\tau}'_j$.

a) pick a random group element $M_i \in \mathbb{G}$ and compute the ElGamal ciphertext $(B_{i,1}, B_{i,2}) \leftarrow \mathsf{El.Enc}(M_i, \hat{X})$
b) invoke $(\{C_{\gamma_{j,i}}\}_{j\in S}, \pi_i) \leftarrow \mathsf{ShareDist}(\{ek_j\}_{j\in S}, t)$.
c) broadcast $(\{C_{\gamma_{j,i}}\}_{j\in S}, \pi_i, B_{i,1}, B_{i,2})$.

- Round 2: On receiving π_j from each $\mathcal{P}_j$, verify them and set $S = S \setminus \{j\}$ if it fails. $\mathcal{P}_i$ does
  a) compute $B_1 = \sum_{j\in S} B_{j,1}$, $B_2 = \sum_{j\in S} B_{j,2}$, and run the partial decryption $M_i \leftarrow \mathsf{El.PartDec}((B_1, B_2), \hat{x}_i)$
  b) generate $\psi_i \leftarrow \mathsf{Prove}_{\mathsf{DDH}}((G, \hat{X}_i, B_1, B_2 - M_i), \hat{x}_i)$
  c) generate $C_{\gamma_j} = \oplus_\nu C_{\gamma_{j,\nu}}$ for each $j \in S$
  d) $(\{C_{\hat{\alpha}_{j,i}}, \hat{\beta}_{i,j}\}_{j\in S}, \hat{\tau}'_i) \leftarrow \mathsf{MPMtA}_2(\{C_{\gamma_j}\}_{j\in S}, x_i)$
  e) $\hat{B}_{i,j} = \hat{\beta}_{i,j} G_2$ for each $j \in S$
  f) broadcast $(M_i, \psi_i, \{C_{\hat{\alpha}_{j,i}}, \hat{B}_{i,j}\}_{j\in S}, \hat{\tau}'_i)$.
- Offline Output: Upon receiving $\psi_j, \hat{\tau}'_j$ from each $\mathcal{P}_j$, set $S = S \setminus \{j\}$ if one of them is not valid. $\mathcal{P}_i$ gets $M \leftarrow \mathsf{El.Combine}(\{M_j\}_{j\in S})$ and the output
  a) $\gamma_i \leftarrow \mathsf{Dec}(dk_i, C_{\gamma_i})$
  b) $\{\zeta_{i,j}\}_{j\in S} \leftarrow \mathsf{MPMtAOut}(dk_i, \{\hat{\beta}_{i,j}, C_{\hat{\alpha}_{i,j}}\}_{j\in S})$
  c) $(e, s) \leftarrow \mathsf{H}(M)$ where $\mathsf{H} : \mathbb{G} \to \mathbb{Z}_q^2$ is a random oracle.

Online Phase. In input of the message $\mathbf{m}$, $D = G_1 + sH_1 + \sum_{k\in[\ell]} m_k H_{k+1} \in \mathbb{G}_1$ is generated as a public parameter.

- One-Round Interaction: $\mathcal{P}_i$ does
  a) pick a random $(t-1)$-degree polynomial $f(\cdot)$ with $f(0) = 0$
  b) generate $(x+e)\gamma$'s share: $\{\chi_{i,j} = e\gamma_i + \zeta_{i,j} + f(j)\}_{j\in S}$
  c) generate $R_i = \gamma_i D$ and broadcast $(\{\chi_{i,j}\}_{j\in S}, R_i)$.
- Online Output: On receiving $\{\chi_{j,\nu}\}_{\nu\in S}$ from each $\mathcal{P}_j$, assemble $A = \dfrac{\sum_{j\in S} \lambda_{j,S} R_j}{\sum_{j,\nu\in S} \lambda_{j,S}\lambda_{\nu,S} \cdot \chi_{j,\nu}} \in \mathbb{G}_1$ if $|S| \geq t$. Run $b \leftarrow \mathsf{Verify}(X, (A, e, s), \mathbf{m})$. It distinguishes two cases.
  a) If $b = 1$, output the signature $\sigma = (A, e, s)$.
  b) Else, the parties identify the cheaters:
     i) generate $\Gamma_j = \sum_{\nu\in S} \lambda_{\nu,S}(\chi_{j,\nu} G_2 - \hat{B}_{j,\nu} + \hat{B}_{\nu,j})$
     ii) check $\mathsf{e}(R_j, eG_2 + X) \stackrel{?}{=} \mathsf{e}(D, \Gamma_j)$ and set $S = S \setminus \{j\}$ if it fails.

  If $|S| \geq t$ still holds, re-assemble $A = \dfrac{\sum_{j\in S} \lambda_{j,S} R_j}{\sum_{j,\nu\in S} \lambda_{j,S}\lambda_{\nu,S} \cdot \chi_{j,\nu}}$ and output $\sigma = (A, e, s)$.

Case b) is required only when misbehavior exists.

6. Security Analysis

In this section, we prove the security of our scheme via the real-vs-ideal paradigm and the Universal Composability (UC) framework [19]. Namely, we show that our scheme realizes an idealized version of the threshold signature task. That is, we prove that for every adversary[8] attacking our scheme, there is an ideal-process adversary (referred to as a simulator) that achieves the same goals.

Note that the major technical difference between non-UC and UC frameworks is that the simulator's arsenal of extraction techniques lacks rewinding in the security analysis of the latter. The prohibition of the rewinding technique requires tools that are amenable to so-called online extraction. If the simulator needs to extract the adversary's secrets from the proofs of knowledge, it will require a more complex Fischlin transform [27], [39], which is the non-rewinding version of the Fiat-Shamir transform [38].

In our security proof, our simulators can always extract the adversary's secrets from the CL ciphertexts rather than the ZKPs. All ZKPs in our scheme need not be proofs of knowledge, except the one for relation $\mathcal{R}_{\mathsf{key}}$ during the key generation phase. Thus, our security proof relies on the $\mathcal{F}_{\mathsf{zk}_{\mathsf{key}}}$-hybrid model. For realization, we only need the Fischlin transform for the ZKP of $\mathcal{R}_{\mathsf{key}}$ in the key generation phase.

6.1. Threshold ECDSA Ideal Functionality

We present threshold ECDSA functionality $\mathcal{F}_{\mathsf{ECDSA}}$ in Figure 2, which is borrowed from [36], [37] with added robustness considerations. This functionality ensures synchrony using the framework of Katz et al. [46], meaning it only advances from one round to the next when all honest parties indicate that it should.

6.2. Security Claims

Theorem 2. *Based on the hard subgroup membership (HSM) assumption over the class group $\mathfrak{G}$, the smoothness of the CL encryption scheme, our threshold ECDSA scheme Π in Section 4 UC-realizes $\mathcal{F}_{\mathsf{ECDSA}}$ in the $\mathcal{F}_{\mathsf{zk}_{key}}$-hybrid model when $n \geq 2t-1$.*

Proof. The theorem is equivalent to the following statement. For every malicious adversary $\mathcal{A}$ that statically corrupts up to $t-1$ parties, there exists a simulator $\mathcal{S}^{\mathcal{A}}$ that uses $\mathcal{A}$ as a black box, such that for every environment $\mathcal{Z}$, random tape z, and every polynomial $\mathsf{poly}(\lambda)$ which bounds the number of times $\mathcal{Z}$ invokes any honest party, it holds that

$$\left\{\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda, z)\right\}_{\substack{\lambda\in\mathbb{N}, n\in O(1),\\ z\in\{0,1\}^*}} \approx_s \left\{\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{ECDSA}},\mathcal{S}^{\mathcal{A}},\mathcal{Z}}(\lambda, z)\right\}_{\substack{\lambda\in\mathbb{N}, n\in O(1),\\ z\in\{0,1\}^*}}$$

We begin by constructing the simulator $\mathcal{S}^{\mathcal{A}}$, which is parameterized by $(\mathbb{G}, G, q, n, t)$, then we will analyze that it produces a view for the environment that is indistinguishable from the real world. In the $\mathcal{F}_{\mathsf{zk}_{\mathsf{key}}}$-hybrid model, $\mathcal{S}^{\mathcal{A}}$ will interact with $\mathcal{A}$ on behalf of the ideal functionality $\mathcal{F}_{\mathsf{zk}_{\mathsf{key}}}$ (defined in Appendix A.1).

Key Generation.

1. When the simulation begins, $\mathcal{A}$ announces the identities of up to $t-1$ corrupted parties. Let the indices of these parties be given by S_c.
2. On receiving $(\mathsf{keygen}, sid, i)$ for some $i \in S_h = \hat{S} \setminus S_c$ from $\mathcal{F}_{\mathsf{ECDSA}}$, reply (keygen, sid) on behalf of $\mathcal{P}_j, j \in S_c$.
3. On receiving X as the public key from $\mathcal{F}_{\mathsf{ECDSA}}$, $\mathcal{S}$ simulates *the distributed key generation protocol* for $\mathcal{A}$.

8. In this work, we consider only *static* adversaries, who announce which $t-1$ parties to corrupt at the beginning of the experiment.

• **Key Generation.**

- On receiving (keygen, sid) from some party $\mathcal{P}_i$ for $i \in [n]$ where sid is fresh, send $(\mathsf{keygen}, sid, i)$ to $\mathcal{S}$.
- After receiving (keygen, sid) from all parties $\mathcal{P}_1, \cdots, \mathcal{P}_n$, if sid is agreed-upon, then
  1. sample $x \leftarrow\$ \mathbb{Z}_q$, compute $X = xG$ and store $(\mathbb{G}, G, q, x, X)$
  2. send X as the public key to $\mathcal{S}$ and all parties
  3. ignore any further call to keygen.

• **Signing.** The signing participants set is S with $|S| \geq t$.

- On receiving (sign, sid, m) from some party $\mathcal{P}_i$, if keygen was already called and sid is fresh, send $(\mathsf{sign}, sid, m, i)$ to $\mathcal{S}$.
- After receiving (sign, sid, m) from all parties, if sid is agreed-upon, then
  1. Compute an ECDSA signature for the hashed message m:
     i) sample $k \leftarrow\$ \mathbb{Z}_q$ and compute $R = kG$
     ii) $r = R|_{x\text{-axis}}$, $s = \frac{m + rx}{k} \bmod q$.
  2. Send (r, s) to $\mathcal{S}$.
  3. If receiving (fail, sid, j) from $\mathcal{S}$, let $C = C \cup \{j\}$, where C records the exposed misbehaving parties for this signature.
  4. Send (r, s) to all parties and store $(\mathsf{completed}, sid)$ in memory.

Figure 2: Ideal Threshold ECDSA Functionality $\mathcal{F}_{\mathsf{ECDSA}}$

- Round 1: For each $i \in S_h$, run CL's key generation $\mathsf{KGen}(\cdot) \to (ek_i, dk_i)$. Send ek_i to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{zk}_{\mathsf{key}}}$.
- Round 2: Receive $\{ek_j, dk_j\}_{j \in S_c}$ from $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{zk}_{\mathsf{key}}}$. For each honest party $\mathcal{P}_i, i \in S_h$, compute all items as prescribed and hand over the shares-distribution messages to $\mathcal{A}$.
- Round 3: On receiving $\{C_{x_{\nu,j}}, \pi_j\}_{j,\nu \in \hat{S}}$, the simulator $\mathcal{S}$ does:
  i) verify π_j for each $j \in S_c$ and set $\hat{S} = \hat{S} \backslash \{j\}, S_c = S_c \setminus \{j\}$ if it fails;
  ii) for each $j \in S_c$, decrypt $\{C_{x_{j,\nu}}\}_{\nu \in \hat{S}}$ to get $\{x_{j,\nu}\}_{\nu \in \hat{S}}$ with the knowledge of the decryption key dk_j and sum them up $x_j = \sum_{\nu \in \hat{S}} x_{j,\nu}$;
  iii) with $(0, X)$ and $\{(j, x_j G)\}_{j \in S_c}$ as the t points, from which a $(t-1)$-degree polynomial over elliptic-curve group $\mathbb{G}$ can be recovered,
  $$f(z)G = z^{t-1}A_{t-1} + \cdots + zA_1 + X \text{ s.t. } f(j) = x_j;$$
  iv) for each $i \in S_h$, compute $X_i = \sum_{l=1}^{t-1} i^l A_l + X$, and simulate the ZKP π'_i.
- Key Generation Output: Verify π'_j for each $j \in S_c$ and set $\hat{S} = \hat{S} \setminus \{j\}, S_c = S_c \setminus \{j\}$ if it fails. $\mathcal{S}$ outputs $\mathbf{x} = (\{x_j\}_{j \in S_c}, \{dk_j\}_{j \in \hat{S}})$.

Signing.

1. On receiving $(\mathsf{sign}, sid, m, i)$ for some $i \in S_h = S \setminus S_c$ from $\mathcal{F}_{\mathsf{ECDSA}}$, reply (sign, sid, m) on behalf of $\mathcal{P}_j, j \in S_c$.
2. On receiving (r, s) from $\mathcal{F}_{\mathsf{ECDSA}}$, $\mathcal{S}$ simulates *the offline phase of threshold signing* with the input of $\mathbf{x}$.
   - Round 1: For each honest party $\mathcal{P}_i, i \in S_h$, pick $\gamma_i \leftarrow\$ \mathbb{Z}_q$, compute all items as prescribed, and hand over the first-round messages to $\mathcal{A}$ on behalf of each honest party.
   - Round 2: On receiving $\{C_{k_{\nu,j}}, \pi_j\}_{j,\nu \in S}$, the simulator $\mathcal{S}$ does:
     i) verify π_j for each $j \in S_c$ and remove j from S, S_c if it fails;
     ii) for each $j \in S_c$, decrypt $\{C_{k_{j,\nu}}\}_{\nu \in S}$ to get $\{k_{j,\nu}\}_{\nu \in S}$ and sum them up $k_j = \sum_{\nu \in S} k_{j,\nu}$;
     iii) with $(0, R)$ and $\{(j, k_j G)\}_{j \in S_c}$ as the t points, from which a $(t-1)$-degree polynomial with each coefficient being an elliptic-curve point can be recovered,
     $$f(z)G = z^{t-1}A_{t-1} + \cdots + zA_1 + R \text{ s.t. } f(j) = k_j;$$
     iv) for each $i \in S_h$, compute $R_i = \sum_{l=1}^{t-1} i^l A_l + R$, and simulate the ZKP π'_i.

   On receiving (C_{γ_j}, τ_j) from each corrupted party $\mathcal{P}_j, j \in S_c$, the simulator $\mathcal{S}$ does:
   i) verify τ_j and remove j from S, S_c if it fails;
   ii) decrypt C_{γ_j} to get γ_j with the knowledge of dk_j;
   iii) for each $i \in S_h$, simulate $\mathcal{P}_i$'s MPMtA_2-message:
   $x_i^*, k_i^* \leftarrow\$ \mathbb{Z}_q$;
   $(\{C_{\alpha_{j,i}}, \beta_{i,j}\}_{j \in S}, \tau'_i) \leftarrow \mathsf{MPMtA}_2(\{C_{\gamma_j}\}_{j \in S}, k_i^*)$;
   $(\{C_{\hat{\alpha}_{j,i}}, \hat{\beta}_{i,j}\}_{j \in S}, \hat{\tau}'_i) \leftarrow \mathsf{MPMtA}_2(\{C_{\gamma_j}\}_{j \in S}, x_i^*)$;
   $\{\alpha_{j,i} = k_i^* \gamma_j - \beta_{i,j}, \hat{\alpha}_{j,i} = x_i^* \gamma_j - \hat{\beta}_{i,j}\}_{j \in S_c}$;
   $\{B_{i,j} = \gamma_j R_i - \alpha_{j,i} G, \hat{B}_{i,j} = \gamma_j X_i - \hat{\alpha}_{j,i} G\}_{j \in S_c}$.
   Hand over the second-round message to $\mathcal{A}$.
   - Offline Output: On receiving the second-round message from each corrupted party $\mathcal{P}_j, j \in S_c$, the simulator $\mathcal{S}$ does:
     i) verify $\mathcal{P}_j$'s ZKPs and remove j from S, S_c if one of them is invalid;
     ii) for each $i \in S_h, j \in S_c$, decrypt $C_{\alpha_{i,j}}, C_{\hat{\alpha}_{i,j}}$ to get $\alpha_{i,j}, \hat{\alpha}_{i,j}$ and compute $\beta_{j,i} = \gamma_i k_j - \alpha_{i,j}$ and $\hat{\beta}_{j,i} = \gamma_i x_j - \hat{\alpha}_{i,j}$ with the knowledge of $\mathcal{P}_j$'s secret share k_j, x_j;
     iii) for each $i \in S_h, j \in S_c$, compute $\delta_{j,i} = \alpha_{j,i} + \beta_{j,i}$ and $\zeta_{j,i} = \hat{\alpha}_{j,i} + \hat{\beta}_{j,i}$;
     iv) output $\mathbf{y} = \{\gamma_j, k_j, \delta_{j,i}, \zeta_{j,i}\}_{j \in S_c, i \in S_h}$.
3. With the input of the hashed message $m \in \mathbb{Z}_q$ and $\mathbf{x}, \mathbf{y}$, $\mathcal{S}$ simulates *the online phase of threshold signing*.
   - One-Round Interaction: Pick $a \leftarrow\$ \mathbb{Z}_q$ and compute $b = s^{-1}(m + r \cdot a)$. Regard a as the dummy signing key and b as the dummy secret nonce. Then $\mathcal{S}$ does:

i) with the knowledge of $\{x_j, k_j\}_{j\in S_c}$, simulate honest parties' shares $\{x_i, k_i\}_{i\in S_h}$ which are t-threshold shares of a and b respectively;
ii) with the knowledge of corrupted parties' shares $\delta_{j,i}, \zeta_{j,i}$, simulate honest parties' shares $\delta_{i,j} = \gamma_j k_i + \gamma_i k_j - \delta_{j,i}$, $\zeta_{i,j} = \gamma_j x_i + \gamma_i x_j - \zeta_{j,i}$ for each $j \in S_c, i \in S_h$;
iii) mask $\delta_{i,j}$ and compute $\chi_{i,j}$ as prescribed for each $j \in S_c, i \in S_h$;
iv) simulate the DDH-ZKP ψ_i for each $i \in S_h$;
v) hand over the online message to $\mathcal{A}$.

- Online Output: check each $\mathcal{P}_j$'s online message with the input of their secret shares, and remove j from S if one of the checks fails. If $|S| \geq t$ still holds, output (r, s).

Analysis. A sequence of hybrid experiments is used to demonstrate indistinguishability between the real and ideal worlds, starting from the real world, which we define first

$$\mathcal{H}_0 = \{\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda, z)\}_{\lambda\in\mathbb{N}, n\in O(1), z\in\{0,1\}^*}.$$

Hybrid $\mathcal{H}_1$. In this hybrid experiment, all honest parties and the ideal functionality $\mathcal{F}_{\mathsf{zk}_{\mathsf{key}}}$ are replaced with $\mathcal{S}$, which plays their roles to interact with the adversary $\mathcal{A}$ and the environment $\mathcal{Z}$. With only syntactical changes, it follows that $\mathcal{H}_1 = \mathcal{H}_0$.

Hybrid $\mathcal{H}_2$. This hybrid behaves identically to $\mathcal{H}_1$, except that the online signing protocol is replaced with a simulated execution. We show that the simulated transcript is statistically indistinguishable from the real one.

In $\mathcal{H}_2$, $\mathcal{S}$ utilizes (r, s) received from $\mathcal{F}_{\mathsf{ECDSA}}$ to simulate honest parties' signature shares. One difference is that $\mathcal{S}$ uses dummy secret key and secret nonce (a, b)'s shares to generate the shares $\delta_{i,j}, \chi_{i,j}$, instead of shares of the correct secret key and secret nonce. The assembled signature with these shares will be $(m+ra)/b$, which is exactly the inputted s. Moreover, the distributions of the simulated shares are identical to those of a real execution. At the same time, the simulated DDH-ZKP ψ_i is indistinguishable from a real one to the environment due to the zero-knowledge property of ZKP. It follows that $\mathcal{H}_2 \approx_s \mathcal{H}_1$.

Hybrid $\mathcal{H}_3$. This hybrid behaves identically to $\mathcal{H}_2$, except that the offline signing protocol is replaced with a simulated execution. We show that it is computationally indistinguishable.

The simulation strategy for the DRG process is identical to that of the key generation phase. We defer this analysis to the next hybrid (i.e., $\mathcal{H}_4$). For MPMtA invocations, the only difference with a real execution is that $\mathcal{S}_2$ uses randomly chosen x_i^*, k_i^* as MPMtA_2's input, instead of correct shares. It is unnoticeable to the environment based on the security of our affine operations (Theorem 1). Moreover, the simulated $B_{i,j}, \hat{B}_{i,j}$ are the same as those in a real execution using the correct x_i and k_i. At the same time, from the zero-knowledge property, the simulated proofs $\tau_i', \hat{\tau}_i'$ are indistinguishable from real ones to the environment. Combined with the computational indistinguishability of the simulated DRG process analyzed in the next hybrid, we have $\mathcal{H}_3 \approx_c \mathcal{H}_2$.

Hybrid $\mathcal{H}_4$. This hybrid experiment replaces the real execution of the distributed key generation protocol with the simulated execution as above. We show that it is computationally indistinguishable.

$\mathcal{S}$ gets X as public key from $\mathcal{F}_{\mathsf{ECDSA}}$. The above simulation ensures that the ECDSA public key generated by DKG equals X. The main difference with a real execution is that the discrete logarithm (DL) of X_i simulated in Round 3 will not be the plaintext of the ciphertext $C_{x_i^*} = \oplus_{\nu\in\hat{S}} C_{x_{i,\nu}}$. The primary distinction between our proof strategy and that of the Paillier-based scheme [20] lies in our simulator having access to the decryption keys of the honest parties. Similar to [22], [23], [24], [59], the CL encryption scheme we employ is based on projective hash functions, which offer statistical security. Therefore, the use of the honest parties' decryption keys does not undermine security, allowing us to reduce the protocol's security to the smoothness of the CL scheme. In conclusion, if the CL encryption scheme is δ_s-smooth and the hard subgroup membership (HSM) problem is δ_{HSM}-hard, then no PPT adversary can notice the plaintext of $C_{x_i^*}$ is not the DL of X_i with probability greater than $2\delta_{\mathsf{HSM}} + 4\delta_s + 3/q$. At the same time, the simulated proof π_i' is indistinguishable from a real one due to the zero-knowledge property of ZKP. Thus, we have $\mathcal{H}_4 \approx_c \mathcal{H}_3$ based on the HSM assumption and the smoothness of CL encryption.

We now have

$$\mathcal{H}_4 = \left\{\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{ECDSA}},\mathcal{S}^{\mathcal{A}},\mathcal{Z}}(\lambda, z)\right\}_{\lambda\in\mathbb{N}, n\in O(1), z\in\{0,1\}^*}$$

and by transitivity, we also have

$$\left\{\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda, z)\right\}_{\substack{\lambda\in\mathbb{N}, n\in O(1),\\ z\in\{0,1\}^*}} \approx_c \left\{\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{ECDSA}},\mathcal{S}^{\mathcal{A}},\mathcal{Z}}(\lambda, z)\right\}_{\substack{\lambda\in\mathbb{N}, n\in O(1),\\ z\in\{0,1\}^*}}$$

based on the HSM assumption and the smoothness of CL encryption. □

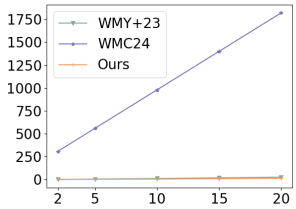
Please refer to Appendix C for the formal definitions of the underlying assumptions — HSM and the smoothness of CL encryption.

7. Performance

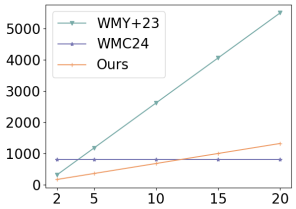
Testbed Environment. Our implementation is done using `BICYCL`[9] [13], an open-source C++ library for class-group arithmetic. We use `mcl`[10] library to implement a pairing-based bilinear group. Experiments were run on a MacBook Pro with macOS Monterey version 12.3, 16GB RAM, and an Apple M1 Pro chip. Our implementation is available at https://github.com/Victoriatang/bicycl_Robust_TECDSA.

Parameters. The computational security level is set to be $\lambda = 128$ bits and statistical security parameter is $\lambda_s = 40$ bits. We use SHA256 to instantiate hash functions. ECDSA

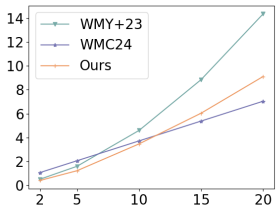
9. https://gite.lirmm.fr/crypto/bicycl
10. https://github.com/herumi/mcl



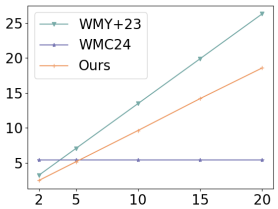
(a) Online time (ms)



(b) Online comm. (bytes)

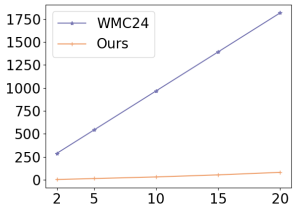


(c) Overall time (s)

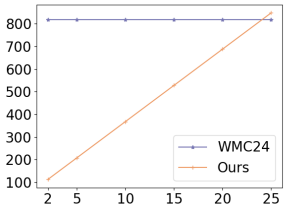


(d) Overall comm. (KB)

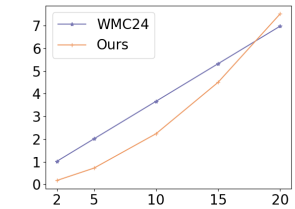
Figure 3: Comparisons of robust threshold ECDSA for the number of parties $n = 2$ to 20.



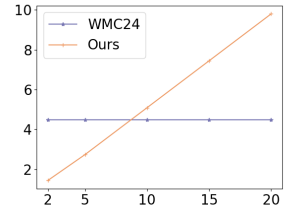
(a) Online time (ms)



(b) Online comm. (bytes)



(c) Overall time (s)



(d) Overall comm. (KB)

Figure 4: Comparisons of robust threshold BBS+

was taken over the elliptic curve (EC) secp256k1, of which the group order is the 256-bit prime q. BBS+ was taken over the pairing curve BLS12_381, of which the group order also has 256 bits. For CL encryption, we use the class group $\mathsf{CL\text{-}HSM}_q$ with 256-bit plaintext space $\mathbb{Z}_q$ and 1827-bit Δ_K.

Asymptotic Analysis. Our protocol has $O(n)$ sending communication and $O(n^2)$ computation (with the quadratic growth driven by CI), like all MtA-based constructions with identifiable abort or robustness [20], [30], [59]. Specifically, compared with the robust scheme [59], our protocol omits the t-thresholding γ step in the offline phase and has a more streamlined online phase. Thus, our scheme can outperform WMY$^+$23 in both communication and computation.

On the other hand, WMC24 [58] has constant sending communication and linear running time for each party. Therefore, our overall efficiency is superior to [58] only in small-scale settings.

We provide benchmarking comparisons with the previous schemes [36][11], [20][12], [58], [59].

Comparisons with robust schemes [58], [59] Table 4 illustrates that our scheme enhances computational efficiency in the online phase by a factor of 2.44 to 2.79, while bandwidth is improved by a factor of 2 to 3. In the offline phase, we enhance approximately $1.3\times$ in both runtime and bandwidth. Figure 3 shows that our scheme consistently outperforms WMY$^+$23 as the number of parties increases. Additionally, when accounting for network latency, the improvement factor increases further due to reducing rounds from 5 to 3.

Table 4 and Figure 3a show that our online phase is extremely fast compared to WMC24, even in large-scale scenarios. As long as the number of parties does not exceed 13, our online communication efficiency is also superior. On the other hand, Figure 3c and 3d show that WMC24's total (offline + online) computational and bandwidth efficiency surpasses ours when $n > 12$ and $n > 5$, respectively. We can draw similar conclusions for the threshold BBS+ schemes based on Table 5 and Figure 4.

Comparisons with non-robust schemes [20], [30], [36]. CGG$^+$20 [20], based on Paillier, can achieve identifiable abort. Its online computational cost is significantly higher, even higher than WMC24. However, the online communication is optimal. Both its offline computation and communication efficiencies are lower than ours, with its communication overhead being three times greater than ours.

DKLs24 [36], based on OT, can only achieve security with abort. Its online computation is optimal, matching that of a single signature verification. Additionally, the offline phase is also very fast. However, its offline communication is significantly higher than ours and others.

CDKS24 [30] adds cheater identification (CI) to DKLs24, enabling identifiable abort. Since there is currently no open-source code, we provide a theoretical analysis. Their online CI process involves two Pedersen-commitment-related ZKPs and two discrete-logarithm-related ZKPs, which is approximately twice as costly as ours. On the other hand, their offline phase performs similarly to DKLs24. As a result, it is much faster than ours, but the bandwidth requirements are significantly higher.

11. https://github.com/silence-laboratories/silent-shard-dkls23-ll
12. https://github.com/ZenGo-X/multi-party-ecdsa

Acknowledgments

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

References

[1] Binance. https://www.binance.com.

TABLE 4: Threshold ECDSA's online and offline costs, with running time (ms) and sending communication (kilobytes) per party for a varying number of parties.

Schemes		2		4		5		10		15		20	
		Time	Comm.	Time	Comm.	Time	Comm.	Time	Comm.	Time	Comm.	Time	Comm.
online	DKLs24 [36]	0.1	0.06	0.1	0.06	0.1	0.06	0.12	0.06	0.14	0.06	0.15	0.06
	CGG+20 [20]	502	0.03	638	0.03	775	0.03	1450	0.03	2282	0.03	2966	0.03
	WMY+23 [59]	1.22	0.32	2.18	0.6	3.18	0.9	11.3	2.6	18.3	4.1	26.3	5.5
	WMC24 [58]	308	0.8	392	0.8	476	0.8	890	0.8	1400	0.8	1820	0.8
	Ours	0.5	0.2	0.78	0.24	1.18	0.3	5.1	0.7	9.1	1.0	14.1	1.32
offline	DKLs24 [36]	4	53	9	159	10.9	212	25.4	477	39.1	743	50.7	1008
	CGG+20 [20]	830	9	1338	13.5	2175	18.6	7518	35	14465	46.4	23452	63
	WMY+23 [59]	506	2.92	816	4.5	1326	6.2	4584	11.4	8820	15.8	14300	21.2
	WMC24 [58]	1023	4.6	1303	4.6	1584	4.6	2740	4.6	3980	4.6	5220	4.6
	Ours	406	2.3	780	3.6	1224	4.9	3460	9.0	6030	13.1	9080	17.2

TABLE 5: Threshold BBS+'s online and offline costs, with running time (ms) and sending communication (kilobytes) per party for a varying number of parties.

Schemes		2		4		5	
		Time	Comm.	Time	Comm.	Time	Comm.
online	[58]	287	0.8	372	0.8	457	0.8
	Ours	2.48	0.11	4.78	0.14	7.28	0.17
offline	[58]	979	3.6	1225	3.6	1471	3.6
	Ours	174	1.3	327	1.7	510	2.2

[2] BitGo. https://www.bitgo.com.

[3] Coinbase. https://www.coinbase.com.

[4] Fireblocks. https://www.fireblocks.com.

[5] Zengo. https://zengo.com.

[6] Damiano Abram, Ariel Nof, Claudio Orlandi, Peter Scholl, and Omer Shlomovits. Low-bandwidth threshold ECDSA via pseudorandom correlation generators. In *S&P*, pages 2554–2572. IEEE, 2022.

[7] Man Ho Au, Willy Susilo, Yi Mu, and Sherman S. M. Chow. Constant-size dynamic k-times anonymous authentication. *IEEE Syst. J.*, 7(2):249–261, 2013.

[8] Carsten Baum, Nikolas Melissaris, Rahul Rachuri, and Peter Scholl. Cheater identification on a budget: MPC with identifiable abort from pairwise macs. *IACR Cryptol. ePrint Arch.*, page 1548, 2023.

[9] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, volume 576, pages 420–432. Springer, 1991.

[10] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS*, pages 62–73. ACM, 1993.

[11] Dan Boneh, Rosario Gennaro, and Steven Goldfeder. Using level-1 homomorphic encryption to improve threshold DSA signatures for bitcoin wallet security. In *LATINCRYPT*, volume 11368, pages 352–377. Springer, 2017.

[12] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In *CRYPTO*, volume 10991, pages 565–596. Springer, 2018.

[13] Cyril Bouvier, Guilhem Castagnos, Laurent Imbert, and Fabien Laguillaumie. I want to ride my BICYCL : BICYCL implements cryptography in class groups. *J. Cryptol.*, 36(3):17, 2023.

[14] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent ot extension and more. In *CRYPTO*, pages 489–518, 2019.

[15] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-lpn. In *CRYPTO*, pages 387–416, 2020.

[16] Luís TAN Brandão and René Peralta. Nist first call for multi-party threshold schemes. https://nvlpubs.nist.gov/nistpubs/ir/2023/NIST.IR.8214C.ipd.pdf, 2023.

[17] Lennart Braun, Ivan Damgård, and Claudio Orlandi. Secure multiparty computation from threshold encryption based on class groups. In *CRYPTO*, pages 613–645. Springer, 2023.

[18] Lennart Braun, Ivan Damgård, and Claudio Orlandi. Secure multiparty computation from threshold encryption based on class groups. In *CRYPTO*, volume 14081, pages 613–645. Springer, 2023.

[19] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145. IEEE Computer Society, 2001.

[20] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In *CCS*, pages 1769–1787. ACM, 2020.

[21] Ignacio Cascudo and Bernardo David. Publicly verifiable secret sharing over class groups and applications to DKG and YOSO. In *EUROCRYPT*, volume 14655, pages 216–248. Springer, 2024.

[22] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Two-party ECDSA from hash proof systems and efficient instantiations. In *CRYPTO*, volume 11694, pages 191–221. Springer, 2019.

[23] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold EC-DSA. In *PKC*, volume 12111, pages 266–296. Springer, 2020.

[24] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold EC-DSA revisited: Online/offline extensions, identifiable aborts proactive and adaptive security. *Theor. Comput. Sci.*, 939:78–104, 2023.

[25] Guilhem Castagnos and Fabien Laguillaumie. Linearly homomorphic encryption from DDH. In *CT-RSA*, volume 9048, pages 487–505. Springer, 2015.

[26] Guilhem Castagnos, Fabien Laguillaumie, and Ida Tucker. Practical fully secure unrestricted inner product functional encryption modulo p. In *ASIACRYPT*, volume 11273, pages 733–764. Springer, 2018.

[27] Yi-Hsiu Chen and Yehuda Lindell. Optimizing and implementing fischlin's transform for uc-secure zero-knowledge. *Cryptology ePrint Archive*, 2024.

[28] Sherman S. M. Chow, Jack P. K. Ma, and Tsz Hon Yuen. Scored anonymous credentials. In *ACNS*, volume 13906, pages 484–515. Springer, 2023.

[29] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *STOC*, pages 364–369. ACM, 1986.

[30] Ran Cohen, Jack Doerner, Yashvanth Kondi, and Abhi Shelat. Secure multiparty computation with identifiable abort from vindicating release. *IACR Cryptol. ePrint Arch.*, page 1136, 2023.

[31] Ran Cohen, Jack Doerner, Eysa Lee, Anna Lysyanskaya, and Lawrence Roy. An unstoppable ideal functionality for signatures and a modular analysis of the dolev-strong broadcast. *Cryptology ePrint Archive*, 2024.

[32] William M Daley and Raymond G Kammer. Digital signature standard (DSS). *BOOZ-ALLEN AND HAMILTON INC MCLEAN VA*, 2000.

[33] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In *CRYPTO*, pages 307–315. Springer, 1989.

[34] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Secure two-party threshold ECDSA from ECDSA assumptions. In *S&P*, pages 980–997. IEEE Computer Society, 2018.

[35] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Threshold ECDSA from ECDSA assumptions: The multiparty case. In *S&P*, pages 1051–1066. IEEE, 2019.

[36] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Threshold ECDSA in three rounds. In *S&P*. IEEE Computer Society, 2024.

[37] Jack Doerner, Yashvanth Kondi, Eysa Lee, Abhi Shelat, and LaKyah Tyner. Threshold BBS+ signatures for distributed anonymous credential issuance. In *S&P*, pages 773–789. IEEE, 2023.

[38] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, volume 263, pages 186–194. Springer, 1986.

[39] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In *CRYPTO*, volume 3621, pages 152–168. Springer, 2005.

[40] Adam Gagol, Jedrzej Kula, Damian Straszak, and Michal Swietek. Threshold ECDSA for decentralized asset custody. *IACR Cryptol. ePrint Arch.*, page 498, 2020.

[41] Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In *CCS*, pages 1179–1194. ACM, 2018.

[42] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In *ACNS*, volume 9696, pages 156–174. Springer, 2016.

[43] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold DSS signatures. In *EUROCRYPT*, volume 1070, pages 354–371. Springer, 1996.

[44] Jens Groth and Victor Shoup. Design and analysis of a distributed ECDSA signing service. *IACR Cryptol. ePrint Arch.*, page 506, 2022.

[45] Aniket Kate, Easwar Vivek Mangipudi, Pratyay Mukherjee, Hamza Saleem, and Sri Aravinda Krishnan Thyagarajan. Non-interactive VSS using class groups and application to DKG. In *CCS*. ACM, 2024.

[46] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In *TCC*, volume 7785, pages 477–498. Springer, 2013.

[47] Yehuda Lindell. Fast secure two-party ECDSA signing. In *CRYPTO*, volume 10402, pages 613–644. Springer, 2017.

[48] Yehuda Lindell and Ariel Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In *CCS*, pages 1837–1854. ACM, 2018.

[49] Yingjie Lyu, Zengpeng Li, Hong-Sheng Zhou, Haiyang Xue, Mei Wang, Shuchao Wang, and Mengling Liu. Bandwidth-efficient robust threshold ecdsa in three rounds. *Cryptology ePrint Archive*, 2025.

[50] Jack P. K. Ma and Sherman S. M. Chow. SMART credentials in the multi-queue of slackness (or secure management of anonymous reputation traits without global halting). In *EuroS&P*, pages 896–912. IEEE, 2023.

[51] Philip D. MacKenzie and Michael K. Reiter. Two-party generation of DSA signatures. In *CRYPTO*, volume 2139, pages 137–154. Springer, 2001.

[52] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO*, volume 435, pages 239–252. Springer, 1989.

[53] Victor Shoup. Practical threshold signatures. In *EUROCRYPT*, volume 1807, pages 207–220. Springer, 2000.

[54] Alberto Sonnino, Mustafa Al-Bassam, Shehar Bano, Sarah Meiklejohn, and George Danezis. Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers. In *NDSS*. The Internet Society, 2019.

[55] Guofeng Tang, Shuai Han, Li Lin, Changzheng Wei, and Ying Yan. Batch range proof: How to make threshold ECDSA more efficient. In *CCS*. ACM, 2024.

[56] Guofeng Tang and Haiyang Xue. Robust Threshold ECDSA with Online-Friendly Design in Three Rounds . In *S&P*, pages 203–221. IEEE Computer Society, 2025.

[57] Dmytro Tymokhanov and Omer Shlomovits. Alpha-rays: Key extraction attacks on threshold ECDSA implementations. *IACR Cryptol. ePrint Arch.*, page 1621, 2021.

[58] Harry W. H. Wong, Jack P. K. Ma, and Sherman S. M. Chow. Secure multiparty computation of threshold signatures made more efficient. In *NDSS*. The Internet Society, 2024.

[59] Harry W. H. Wong, Jack P. K. Ma, Hoover H. F. Yin, and Sherman S. M. Chow. Real threshold ECDSA. In *NDSS*. The Internet Society, 2023.

[60] Haiyang Xue, Man Ho Au, Mengling Liu, Kwan Yin Chan, Handong Cui, Xiang Xie, Tsz Hon Yuen, and Chengru Zhang. Efficient multiplicative-to-additive function from joye-libert cryptosystem and its application to threshold ECDSA. In *CCS*, pages 2974–2988. ACM, 2023.

[61] Haiyang Xue, Man Ho Au, Xiang Xie, Tsz Hon Yuen, and Handong Cui. Efficient online-friendly two-party ECDSA signature. In *CCS*, pages 558–573. ACM, 2021.

[62] Tsz Hon Yuen, Handong Cui, and Xiang Xie. Compact zero-knowledge proofs for threshold ECDSA with trustless setup. In *PKC*, volume 12710, pages 481–511. Springer, 2021.

Appendix A. Zero-Knowledge Proofs

A.1. Σ-Protocols

Let $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ be a PPT decidable binary relation. We call w a witness for an instance x if $(x, w) \in \mathcal{R}$, and define the language $\mathcal{L} = \{x | \exists\ w,\ \text{s.t. } (x, w) \in \mathcal{R}\}$ as the set of instances x that have a witness w in the relation $\mathcal{R}$. A Σ-protocol for a statement-witness pair (x, w) in a binary relation $\mathcal{R}$ is a three-round protocol between the prover $\mathcal{P}$ and the verifier $\mathcal{V}$.

1) $\mathcal{P}$ takes the state τ as input and outputs the first-round message A.
2) $\mathcal{V}$ sends a random challenge e to prover.
3) $\mathcal{P}$ takes (x, w, τ, e) as input and outputs a response z.

In this paper, we also consider an extended five-round Σ-protocol. Building on the aforementioned three-round protocol, two additional rounds are introduced: $\mathcal{V}$ sends another random challenge e^* after receiving the response z, then $\mathcal{P}$ takes (x, w, τ, e^*) as input and outputs another response z^*.

By the Fiat-Shamir transform [38] or the Fischlin transform [39], an interactive Σ-protocol can be easily converted

to a non-interactive, zero-knowledge argument in the random oracle model [10].

The (non-interactive) zero-knowledge proof system must be equipped with the following properties.

Definition 2 (Completeness)**.** *If $(x, w) \in \mathcal{R}$ and an honest $\mathcal{P}$ with witness w outputs the proof (A, z), then $\mathcal{V}$ can be convinced that x is true by verifying (x, A, e, z).*

Definition 3 (Standard Soundness)**.** *If $x \notin \mathcal{R}$, no PPT prover can output a valid proof (A, z), convincing an honest $\mathcal{V}$ that x is true with non-negligible probability.*

Definition 4 (Honest-Verifier Zero-Knowledge, HVZK)**.** *There exists a PPT simulator $\mathcal{S}$ that takes (x, e) as input, and outputs an accepting transcript (x, A, e, z) indistinguishable from a real one to $\mathcal{V}$.*

A zero-knowledge proof of knowledge is a stronger form of zero-knowledge proof that also satisfies the proof-of-knowledge property. The proof-of-knowledge property is a stronger property than standard soundness.

Definition 5 (Proof of Knowledge)**.** *There exists a PPT extractor $\mathcal{E}$ 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.*

For a relation $\mathcal{R}$, a better way to capture a proof of knowledge is to use the standard ideal zero-knowledge proof-of-knowledge functionality $\mathcal{F}_{\mathsf{zk}_\mathcal{R}}$.

> Functionality $\mathcal{F}_{\mathsf{zk}_\mathcal{R}}$ interacts with all parties as follows: Upon receiving $(\mathsf{prove}, \mathsf{sid}, x, w)$ from party $\mathcal{P}_i$ where $i \in [n]$, $\mathcal{F}_{\mathsf{zk}_\mathcal{R}}$ broadcasts $(\mathsf{proved}, \mathsf{sid}, x, i)$ to all parties if $(x, w) \in \mathcal{R}$. Otherwise, it ignores the message.

If a larger protocol uses a proof of knowledge for relation $\mathcal{R}$, the ideal functionality $\mathcal{F}_{\mathsf{zk}_\mathcal{R}}$ can make the security analysis easier.

A.2. Σ-Protocols for some NP-Relations

In this part, we provide the Σ-protocols for the NP-relations defined in Section 2.3. We remark that only the Σ-protocol for $\mathcal{R}_{\mathsf{key}}$ needs to be a proof of knowledge. The other protocols only need to meet standard soundness. For those protocols, it is not necessary to construct an extractor to explicitly extract the witness; it suffices to demonstrate the witness's existence.

For class-group-based relation $\mathcal{R}_{\mathsf{key}}$, we adapt a 5-round Σ-protocol [62, Algorithm 2] by Yuen et al. Refer to [62] for the proof of its proof-of-knowledge property.

For class-group-based relation $\mathcal{R}_{\mathsf{Dec\text{-}DL}}$, we can rearrange it as

$$\mathcal{R}_{\mathsf{Dec\text{-}DL}} = \left\{ \begin{array}{c} ((C_m = (c_1, c_2), A), \\ (m \in \mathbb{Z}_q, dk \in \mathcal{D})) \end{array} \middle| \begin{array}{c} c_2 = \mathfrak{f}^m c_1^{dk}, \\ ek = \mathfrak{g}_q^{dk}, A = mG \end{array} \right\}.$$

We present 3-round Σ-protocol in Figure 5 for it. Following [18], [58], standard soundness is satisfied with soundness error $2^{-\lambda}$.

Prover(m, dk)		Verifier(C_m, A)
$m_0 \leftarrow\$ \mathbb{Z}_q$,		
$dk_0 \leftarrow\$ [0, 2^{\lambda+\lambda_s}\mathcal{D}]$		
$c_2' = \mathfrak{f}^{m_0} c_1^{dk_0}, ek_0 = \mathfrak{g}_q^{dk_0}$		
$A_0 = m_0 G$	$\xrightarrow{(c_2', ek_0), P_{m_0}}$	$\mathsf{ch} \leftarrow\$ [0, 2^\lambda]$
$m^* = m_0 + \mathsf{ch} \cdot m \in \mathbb{Z}_q$	$\xleftarrow{\mathsf{ch}}$	
$dk^* = dk_0 + \mathsf{ch} \cdot dk \in \mathbb{Z}$	$\xrightarrow{m^*, dk^*}$	check $dk^* \in [0, 2^{\lambda+\lambda_s}\mathcal{D}]$
		$m^* G \stackrel{?}{=} A_0 + \mathsf{ch} \cdot A$
		$\mathfrak{g}_q^{dk^*} \stackrel{?}{=} ek_0 ek^{\mathsf{ch}}$
		$\mathfrak{f}^{m^*} c_1^{dk^*} \stackrel{?}{=} c_2' c_2^{\mathsf{ch}}$

Figure 5: Σ-protocol for $\mathcal{R}_{\mathsf{Dec\text{-}DL}}$ over $(C_m = (c_1, c_2), A \in \mathbb{G})$

With swapping the position of c_1 and ek, $\mathcal{R}_{\mathsf{Dec\text{-}DL}}$ is very similar with the relation

$$\mathcal{R}_{\mathsf{Enc}} = \left\{ \begin{array}{c} ((C_m = (c_1, c_2)), \\ (m \in \mathbb{Z}_q, \rho \in \mathcal{D}_q)) \end{array} \middle| \begin{array}{c} c_2 = \mathfrak{f}^m ek^\rho, \\ c_1 = \mathfrak{g}_q^\rho \end{array} \right\}.$$

Thus, the Σ-protocol for $\mathcal{R}_{\mathsf{Enc}}$ is a simplified variant of that for $\mathcal{R}_{\mathsf{Dec\text{-}DL}}$, without involving the EC group element.

The ZKP for the EC-group-based relation $\mathcal{R}_{\mathsf{DDH}}$ is well-studied [52]. It is much more efficient than others as it only requires operations over the EC group $\mathbb{G}$.

Due to space constraints, we refer the readers to [21, Figure 4] for the concrete construction of ZKP for the class-group-based relation $\mathcal{R}_{\mathsf{Sh}}$.

A.3. ZKP for relation $\mathcal{R}_{\mathsf{m\text{-}AffDL}}$

Given a random oracle $\mathsf{H} : \{0,1\}^* \to [0, 2^\lambda]^{|S|}$, the prover $\mathcal{P}$ and the Verifier $\mathcal{V}$ both do the following

- generate $\{e_j\}_{j \in S} \leftarrow \mathsf{H}(\{C_{\gamma_j}, C_{\alpha_j}, B_j\}_{j \in S}, R)$
- $\mathsf{Aggregate}(\{e_j, C_{\gamma_j}, C_{\alpha_j}, B_j\}_{j \in S}) \to (C_{\gamma^*}, C_{\alpha^*}, B^*)$

Then, $\mathcal{P}$ aggregates the witness: $\beta = \sum_{j \in S} e_j \beta_j \mod q$. It takes (k^*, β) as witness, proves $\mathcal{R}^*$ as in Figure 6, and outputs the proof $((d_1', d_2'), B_0, R_0, \hat{\beta}, \hat{k}^*)$.

After receiving the $\mathcal{R}^*$-proof, $\mathcal{V}$ takes $(C_{\gamma^*}, C_{\alpha^*}, B, R)$ as the statement, verifies the $\mathcal{R}^*$-proof as in Figure 6.

We remark that the ZKP for $\mathcal{R}_{\mathsf{m\text{-}AffDL}}$ needs *not* to be a proof of knowledge. Thus, we only show that it has standard soundness and HVZK property.

Theorem 3. *In the random oracle model, our ZKP for $\mathcal{R}_{\mathsf{m\text{-}AffDL}}$ has standard soundness with soundness error $2^{-\lambda+1}$ and HVZK with statistical distance $2^{-\lambda_s}$.*

Proof. Standard Soundness. Due to the randomness of the challenge $\mathsf{ch} \leftarrow\$ [0, 2^\lambda]$, it is clear that $\mathcal{R}^*$-proof has standard soundness with soundness error $2^{-\lambda}$. So we have that the statement $(C_{\gamma^*}, C_{\alpha^*}, B, R)$ satisfies $\mathcal{R}^*$-relation. Assume that there is some $\bar{j}$ being $C_{\alpha_{\bar{j}}} = (c_{\bar{j},1}^{k'}, c_{\bar{j},2}^{k'} \mathfrak{f}^{-\beta_{\bar{j}}})$ for a different $k' \neq k^*$. In order to pretend that $C_{\alpha_{\bar{j}}}$ is correct,

$\mathcal{P}(k^* \in \mathcal{D}^*, \beta \in \mathbb{Z}_q)$		$\mathcal{V}(C_{\gamma^*}, C_{\alpha^*}, B, R)$
$\beta_0 \leftarrow\$ \mathbb{Z}_q$,		
$k_0^* \leftarrow\$ [0, 2^{\lambda+\lambda_s}\mathcal{D}^*]$		
$d_1' = c_1^{k_0^*}, d_2' = c_2^{k_0^*} \mathfrak{f}^{-\beta_0}$		
$B_0 = \beta_0 G$		
$R_0 = (k_0^* \bmod q)G$	$\xrightarrow{(d_1', d_2'), B_0, R_0}$	$\mathsf{ch} \leftarrow\$ [0, 2^\lambda]$
$\hat{\beta} = \beta_0 + \mathsf{ch} \cdot \beta \in \mathbb{Z}_q$	$\xleftarrow{\mathsf{ch}}$	
$\hat{k}^* = k_0^* + \mathsf{ch} \cdot k^* \in \mathbb{Z}$	$\xrightarrow{\hat{\beta}, \hat{k}^*}$	check $\hat{k}^* \in [0, 2^{\lambda+\lambda_s}\mathcal{D}^*]$
		$c_1^{\hat{k}^*} \stackrel{?}{=} d_1' d_1^{\mathsf{ch}}$
		$c_2^{\hat{k}^*} \mathfrak{f}^{-\hat{\beta}} \stackrel{?}{=} d_2' d_2^{\mathsf{ch}}$
		$(\hat{k}^* \bmod q)G \stackrel{?}{=} R_0 + \mathsf{ch}R$
		$\hat{\beta}G \stackrel{?}{=} B_0 + \mathsf{ch}B$

Figure 6: Σ-protocol for $\mathcal{R}^*$ over $(C_{\gamma^*} = (c_1, c_2), C_{\alpha^*} = (d_1, d_2), (B, R) \in \mathbb{G}^2)$

a malicious prover $\mathcal{P}^*$ needs to eliminate $C_{\alpha_{\tilde{j}}}^{e_{\tilde{j}}}$ successfully when aggregating them into C_{α^*}. In the random oracle model, the probability of $\mathcal{P}^*$ successfully guesses $e_{\tilde{j}}$ is $2^{-\lambda}$. Thus, we conclude that it has standard soundness with soundness error $2^{-\lambda+1}$.

HVZK. As the output is $\mathcal{R}^*$-proof, it suffices to show that $\mathcal{R}^*$-proof is zero-knowledge. It is easy to simulate the proof $((d_1', d_2'), B_0, R_0, \hat{\beta}, \hat{k}^*)$ with the input ch. Pick $\hat{\beta} \leftarrow\$ \mathbb{Z}_q$, and $\hat{k}^* \leftarrow\$ [0, 2^{\lambda+\lambda_s}\mathcal{D}^*]$ whose distribution is $2^{-\lambda_s}$ close to the real distribution. Then compute $(d_1' = c_1^{\hat{k}^*}/d_1^{\mathsf{ch}}, d_2' = c_2^{\hat{k}^*} \mathfrak{f}^{-\hat{\beta}}/d_2^{\mathsf{ch}})$ and $R_0 = (\hat{k}^* \bmod q)G - \mathsf{ch}R, B_0 = \hat{\beta}G - \mathsf{ch}B$. Finally, we conclude that the simulated proof is at a distance $2^{-\lambda_s}$ from the distribution of a real one. □

Appendix B. Security of Multi-Party MtA

In multi-party MtA, the only security concern is about k's safety when the affine algorithm Aff (c.f. Section 3.3) applies the same k for homomorphic operations on different ciphertexts (including ciphertexts that can be decrypted by corrupted parties). For this, we provide a detailed proof of Theorem 1 below.

Proof. For simplicity of exposition, we assume that $|S| = 2$ and $t = 2$ - extending to the general case is straightforward. Suppose that $\mathcal{A}$ corrupts dk_2 and outputs $C_{\gamma_2} = (c_{2,1}, c_{2,2})$ ($\mathcal{A}$ must provide its well-formedness proof simultaneously). Given two choices $k^{(0)}$ and $k^{(1)}$ provided by $\mathcal{A}$, let us call the indistinguishability between Game_0 with $b = 0$ and Game_1 with $b = 1$.

In Game_0 with $k_0^* = k^{(0)} + eq$, the adversary will receive $C_{\alpha_1} = (c_{1,1}^{k_0^*}, c_{1,2}^{k_0^*} \mathfrak{f}^{\beta_1}), C_{\alpha_2} = (c_{2,1}^{k_0^*}, c_{2,2}^{k_0^*} \mathfrak{f}^{\beta_2})$. Since it knows dk_2 and can decrypt C_{α_2}, $\mathcal{A}$ can get $\alpha_2 = k^{(0)}\gamma_2 + \beta_2 \in \mathbb{Z}_q$. Due to $\beta_2 \leftarrow\$ \mathbb{Z}_q$, α_2 is uniformly random. $\mathcal{A}$'s view can be expressed as $(C_{\alpha_1} = (c_{1,1}^{k_0^*}, c_{1,2}^{k_0^*} \mathfrak{f}^{\beta_1}), \mathfrak{g}_q^{rk_0^*}, ek_2^{rk_0^*})$ where r is $\mathcal{A}$'s randomness used in C_{γ_2}.

As we pick $e \leftarrow\$ \mathcal{D}_q$ randomly, the distribution of $\mathfrak{g}_q^{rk_0^*}, ek_2^{rk_0^*}$ is at distance less than $2^{-\lambda_s}$ from the uniform distribution over $\mathfrak{G}^q$.

In Hybrid_1 with $k_1^* = k^{(1)} + eq$, we can write $\mathcal{A}$'s view as $(C_{\alpha_1} = (c_{1,1}^{k_0^*}, c_{1,2}^{k_0^*} \mathfrak{f}^{\beta_1}), \mathfrak{g}_q^{rk_1^*}, ek_2^{rk_1^*})$. It is statistically indistinguishable from Game_0 from the above analysis.

In Hybrid_2, we rewrite $C_{\alpha_1} = (\mathfrak{g}_q^{\bar{r} \cdot k_0^*}, ek_1^{\bar{r} \cdot k_0^*} \mathfrak{f}^{\alpha_1})$ with $\alpha_1 = k^{(0)}\gamma_1 + \beta_1 \bmod q$. From $\mathcal{A}$'s view, C_{α_1} is identically distributed with a fresh CL ciphertext with unknown randomness $\bar{r} \cdot k_0^* \leftarrow\$ \mathcal{D}_q$ and unknown key dk_1.

In Hybrid_3, we can regard C_{α_1} as $(\mathfrak{g}_q^{\bar{r}' \cdot k_1^*}, ek_1^{\bar{r}' \cdot k_1^*} \mathfrak{f}^{\alpha_1})$ with $\bar{r}' \cdot k_1^* = \bar{r} \cdot k_0^*$ and $\alpha_1 = k^{(1)}\gamma_1 + \beta_1'$. Based on the HSM assumption (Definition 6), the CL encryption is semantic secure [25], and thereby Hybrid_3 is indistinguishable from Hybrid_2.

Hybrid_3 is exactly Game_1. Hence, we can conclude that Game_0 and Game_1 are indistinguishable based on the HSM assumption. □

Appendix C. Assumptions over Class Group

Definition 6 (HSM assumption [13], [26])**.** *For the class group $\mathfrak{G} = \mathfrak{F} \times \mathfrak{G}^q$ (c.f. Section 2.2), let $\mathcal{A}$ be an adversary for the HSM problem, and its advantage is defined as*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{HSM}}(\lambda) := \left| 2 \cdot \Pr \left[\begin{array}{c} b = b^* : x \leftarrow\$ \mathcal{D}, u \leftarrow\$ \mathbb{Z}_q, \\ b \leftarrow\$ \{0,1\}, z_0 = \mathfrak{f}^u \mathfrak{g}_q^x, z_1 = \mathfrak{g}_q^x, \\ b^* \leftarrow \mathcal{A}(\mathfrak{G}, z_b, \mathsf{Dlog}(\cdot)) \end{array} \right] - \frac{1}{2} \right|.$$

The HSM problem is δ_{HSM}-hard if for any PPT adversary $\mathcal{A}$, its advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{HSM}}(\lambda) \leq \delta_{\mathsf{HSM}}$. The HSM assumption holds if δ_{HSM} is negligible.

Lemma 1. *If the uniform distribution over $\mathcal{D}$ is δ-close to the uniform distribution over $\mathbb{Z}_{q\tilde{s}}$, where $q\tilde{s}$ is an upper bound of the order of the unknown-order class group $\mathfrak{G}$. For any $\mathfrak{g} \in \mathfrak{G} \setminus \mathfrak{G}^q$, $\mathfrak{f}^m \in \mathfrak{F}$ where $m \leftarrow\$ \mathbb{Z}_q$ and $x \leftarrow\$ \mathcal{D}$, the distributions $\mathcal{D}_1 := \{\mathfrak{g}, \mathfrak{f}^m \cdot \mathfrak{g}^x\}$ and $\mathcal{D}_2 := \{\mathfrak{g}, \mathfrak{g}^x\}$ are 2δ-close.*

The proof of the above lemma can be found in [22].

Definition 7 (Smoothness [23])**.** *The CL encryption scheme is said to be δ_s-smooth if the uniform distribution over $\mathcal{D}$ from which secret keys are sampled is δ-close to the uniform distribution over $\mathbb{Z}_{q\tilde{s}}$, where $\delta_s = 2\delta$.*