Online-Friendly Robust Threshold ECDSA with Constant Amortized Communication

Guofeng Tang
Singapore Management University
tang.guofeng789@gmail.com

Tian Qiu
Nanyang Technological University
qtautumn6@gmail.com

Bowen Jiang
Singapore Management University
bowen.jiang.2024@phdcs.smu.edu.sg

Haiyang Xue
Singapore Management University
haiyangxc@gmail.com

Meng Hao
Singapore Management University
menghao303@gmail.com

Guomin Yang
Singapore Management University
gmyang@smu.edu.sg

Robert H. Deng
Singapore Management University
robertdeng@smu.edu.sg

***Abstract*—Threshold ECDSA has been an active research topic in recent years, driven by its wide-ranging applications, particularly in blockchain domains. Existing constructions of threshold ECDSA generally fall into two categories: those based on threshold linearly homomorphic encryption (TLHE) and those leveraging the Multiplicative-to-Additive (MtA) paradigm. The TLHE-based approach (e.g., WMC24 in NDSS'24) achieves constant communication per party but incurs an expensive online phase and requires a broadcast channel. In contrast, the MtA-based approach (e.g., DKLs24 in S&P'24) offers an optimal online phase and avoids the use of a broadcast channel. However, it has the drawback of requiring linear (i.e., $O(n)$) communication per party when n parties are involved.**

In this work, we propose an MtA-based threshold ECDSA scheme with constant amortized communication. At the core of our approach is the use of packed secret sharing, which enables the same MtA operations to generate $\ell = \epsilon n$ signatures. With a constant ϵ, the communication complexity per signature is amortized to a constant in dishonest-majority settings. Furthermore, we extend this packing technique to design a robust threshold ECDSA with constant communication under honest-majority settings, which ensures the delivery of valid signatures as long as a sufficient number of parties are honest. In contrast, the state-of-the-art robust MtA-based construction (TX25 in S&P'25) requires linear communication per party. We implement our packed constructions using both CL-based and OT-based MtA protocols. Benchmark results show that our amortized efficiency surpasses that of DKLs24. Moreover, our robust scheme outperforms TX25 and has significantly better online efficiency with comparable overall complexity to WMC24.

1. Introduction

A (t, n) threshold signature scheme divides the private signing key into n shares where n is the total number of parties, such that any group of $t+1$ parties can collaborate to generate a signature without revealing the signing key, while any group of t (or less) parties cannot generate a signature [56]. Threshold signatures have gained significant attention in the context of cryptocurrencies, as they enable secure management of keys that authorize cryptocurrency transactions. The importance of threshold signature (and multi-party threshold cryptography in general) is acknowledged by NIST in its upcoming standardization process [12].

Since ECDSA is currently the most widely used signature scheme in cryptocurrencies, many academic studies have focused on the efficient design of threshold ECDSA [30], [32], [38], [49], [51]. Additionally, several "wallet-as-a-service" (WaaS) providers, such as Fireblocks, Coinbase, Binance, and Dfns, have adopted these schemes to secure cryptocurrency wallets for both financial institutions and end-user consumers.

Recall that an ECDSA signature on a message msg is

$$(r = kG|_{x\text{-axis}}, s = \frac{m + rx}{k} \bmod q)$$

where G is the generator of an elliptic curve (EC) group with order q, x is the secret key, k is the secret nonce, r is the x-coordinate of kG, $m = \mathsf{H}(\mathsf{msg})$ and $\mathsf{H}(\cdot)$ is a hash function. To enable distributed signing, the private values x and k are both shared among multiple participants, with each party P_i receiving a share (x_i, k_i). However, an obstacle arises when attempting to jointly compute the multiplicative (non-linear) components k^{-1} and $k^{-1} \cdot x$ with (x_i, k_i) as each party's input.

This obstacle can be addressed using different approaches depending on whether the majority is honest or

dishonest. In an honest majority setting, we set $t < \frac{n}{2}$, while in the dishonest majority setting, we assume an adversary that corrupts $\frac{n}{2} \leq t < n$ out of n parties.

Honest-Majority Threshold ECDSAs. In the honest-majority setting, threshold ECDSA [28], [48] can be constructed solely based on Shamir secret sharing. Roughly speaking, given secrets a and b, let $[a]_t, [b]_t$ denote degree-t Shamir shares of a, b respectively [55]. Shamir secret sharing supports a multiplicative operation that yields a degree-$2t$ sharing of the product $[ab]_{2t} = [a]_t \cdot [b]_t$ when $t < \frac{n}{2}$, where each party locally computes the product of its shares [29]. In threshold ECDSA, computing k^{-1} or $k^{-1}x$ follows a similar apporach: given $[k]_t$ and another shared secret nonce $[\gamma]_t$, we first compute $[k\gamma]_{2t}$ and then derive $[k^{-1}]_t = (k\gamma)^{-1}[\gamma]_t$ by assembling $2t+1$ shares of $k\gamma$.

As shown by Katz et al. [48] (denoted by KU24), in this setting, we can achieve a highly efficient threshold ECDSA protocol with $O(1)$ sending communication per party and $O(n)$ computational complexity per party.

Dishonest-Majority Threshold ECDSAs. In the dishonest-majority setting, threshold ECDSA requires more advanced cryptographic tools and can be classified into two categories: those based on threshold homomorphic encryption and those relying on multiplicative-to-additive conversion.

Threshold Linearly Homomorphic Encryption. Linearly homomorphic encryption (LHE) is an encryption scheme such that under a public key pk, given encryptions of a, b, denoted as $\mathsf{Enc}(pk, a)$ and $\mathsf{Enc}(pk, b)$, we have $c \odot \mathsf{Enc}(pk, a) \oplus \mathsf{Enc}(pk, b) = \mathsf{Enc}(pk, ac + b)$ for a constant c. In threshold LHE (TLHE), the secret decryption key is distributed among multiple parties (using an appropriate secret-sharing scheme). A secure distributed decryption protocol ensures the recovery of the plaintext without anything else. Braun et al. [13] demonstrated that, assuming the availability of a broadcast channel[1], TLHE can support general multi-party computation (MPC) in a "You-Only-Speak-Once" (YOSO) manner when combined with appropriate zero-knowledge proofs (ZKPs). Recently, Wong et al. [59] extended this approach to support threshold ECDSA functionality (denoted by WMC24 hereafter).

WMC24 also benefits from $O(1)$ sending communication per party and $O(n)$ computational complexity. However, it relies on the relatively expensive broadcast channel, and requires costly online operations[2].

Multiplicative-to-Additive (MtA) Conversion. Another approach for threshold ECDSA in the dishonest-majority setting is MtA conversion [14], [32], [38], [49], [57]. In essence, MtA is a two-party protocol that, given inputs a and b from each party, outputs additive shares α and β, such that $\alpha + \beta = a \cdot b$. MtA can be efficiently constructed from homomorphic encryptions[3] such as Paillier [38], [51], Castagnos-Laguillaumie (CL) [18], [20], Joye-Libert (JL) [61], and oblivious transfer (OT) [30], [31].

As shown in Table 1, MtA-based approach in [32] (denoted by DKLs24 hereafter) offers an extremely efficient online phase compared to TLHE-based scheme [59], which is referred to as *online-friendly* [58]. It also does not rely on a heavy broadcast channel. However, the approach requires executing MtAs between each pair of parties, resulting in $O(n)$ sending communication overhead and $O(n)$ computational complexity per party.

Robustness. An important requirement for threshold ECDSA in real-world applications is robustness (also known as guaranteed output delivery in generic MPC [2]), which ensures that the protocol always produces a valid signature as long as a sufficient number of parties remain honest. Achieving this enhanced notion of security requires a broadcast channel [24], and an honest-majority setting ($n \geq 2t + 1$) since $t + 1$ honest parties are necessary to contribute correct shares [23], [50].

Establishing robustness for Shamir-based threshold ECDSA, which uses $[ab]_{2t} = [a]_t \cdot [b]_t$, would need a more stringent requirement ($n \geq 3t + 1$), as shown in [39], [43]. We do not delve into the details of these protocols since our work aims to achieve "best-of-both-worlds" security: it is secure under the dishonest-majority setting, and provides robustness under the honest-majority setting.

On the other hand, there are well-established techniques for enhancing robustness on both TLHE-based (as in WMC24) and MtA-based schemes (as in [58], denoted by TX25) when $n \geq 2t+1$. Their asymptotic performance differs. The robust TLHE-based scheme, WMC24, maintains $O(1)$ communication and $O(n)$ computation, whereas the MtA-based scheme, TX25, incurs an $O(n)$ communication and $O(n^2)$ computation per party.

Motivation. As analyzed above and summarized in Table 1, MtA-based constructions require linear communication complexity per party, regardless of whether robustness is considered, whereas other approaches achieve constant communication complexity per party. Despite its asymptotic disadvantages, the MtA method is round-minimal and enables the design of an optimal online phase in the dishonest-majority setting [32], and unlike TLHE-based schemes [59], it does not require a broadcast channel. These advantages make it particularly appealing for real-world applications.

Given the current state of the art, a natural question arises: *is it possible to reduce the communication complexity of MtA-based schemes to $O(1)$ per party, thereby making them competitive with TLHE-based schemes in terms of asymptotic performance?*

This work addresses the motivation question through the following contributions.

1. A broadcast channel is a communication model in which a sender transmits the same message to all recipients simultaneously. It is generally considered more expensive than a point-to-point channel.

2. Threshold signing protocols can be divided into offline (i.e., preprocessing) and online phases depending on whether the signing message is required.

3. Unlike the TLHE-based scheme, MtA relies solely on the linear homomorphic property but does not require threshold decryption.

TABLE 1: Asymptotic performance (sending communication and computation per party) of threshold ECDSAs

Schemes	Type	Comm. Cost	Comp. Cost	Rounds	Online-Friendly°	Broadcast#	Robustness	Corr. Bound
KU24 [48]	Shamir-based	$O(1)$	$O(n)$	2	✓	no	×	$t < \frac{n}{2}$
WMC24 [59]	TLHE-based	$O(1)$	$O(n)$	4	×	yes	×	$t < n$
		$O(1)$	$O(n)$		×	yes	✓	$t < \frac{n}{2}$
DKLs24 [32]†	MtA-based	$O(n)$	$O(n)$	3	✓	no	×	$t < n$
TX25 [58]		$O(n)$	$O(n^2)$		✓	yes	✓	$t < \frac{n}{2}$
Ours	MtA-based	$\frac{O(n)}{\epsilon n} \approx O(1)$	$\frac{O(n^2)}{\epsilon n} \approx O(n)$	3	✓	no	×	$t \leq n(1-\epsilon)^\star$
		$\frac{O(n)}{n-2t} \approx O(1)$	$\frac{O(n^2)}{n-2t} \approx O(n)$		✓	yes	✓	$t < \frac{n}{2}$

° If the online computation requires only EC group operations, then this scheme is online-friendly.
"yes" indicates that the protocol assumes a broadcast channel. "no" indicates it only needs point-to-point channels.
† This row (except for the "Round" column) also applies to other MtA-based schemes, such as CGG$^+$20 [14], CCL$^+$20 [19], etc.
⋆ $0 < \epsilon < 1$. When $0 < \epsilon < \frac{1}{2}$, $t \leq n(1-\epsilon)$ encompasses dishonest-majority settings. Here, the packing parameter ℓ is set as $\ell = \epsilon n$. Our asymptotic performance is amortized over ϵn signatures.

1.1. Our Contributions

We present an MtA-based threshold ECDSA construction that achieves $O(1)$ amortized sending communication complexity per party. Table 1 gives a detailed comparison. Our construction leverages packed secret sharing, where a single Shamir's share can embed multiple secret values [37]. Our core improvement lies in the MtA amortization: while each party still performs $O(n)$ MtA calls as in prior MtA-based works [14], [32], these calls can now be used to generate $\ell = \epsilon n$ signatures (instead of one signature), where ℓ is the packing parameter, $0 < \epsilon < 1$ (typically $0 < \epsilon < \frac{1}{2}$ in the dishonest-majority settings) and $n \geq t+\ell$. As a result, the total cost is amortized among ϵn signatures. We should emphasize that the amortization only occurs in the offline phase, and the online phase can handle each message to be signed independently.

We extend our packed scheme to support robustness while maintaining $O(1)$ amortized communication. The offline phase is achieved by using a strong packed distributed randomness generation (sPDRG) and MtA with public checking incorporating zero-knowledge proofs as in [58]. The online phase presents a greater challenge: while previous non-robust MtA schemes naturally achieve constant online communication, robust schemes [58], [60] inherently require at least linear communication overhead. To overcome this, we embed ℓ messages into the original signature shares, enabling the batched generation of $\ell = n - 2t$ signatures, thereby amortizing the online communication cost to a constant. We note that this online phase approach requires ℓ messages to be signed simultaneously.

We implement our packed threshold ECDSA scheme using both CL-based MtA and OT-based MtA in the dishonest-majority setting and compare our scheme against the state-of-the-art MtA-based protocol DKLs24 [32]. While our non-packed version (i.e., when $\ell = 1$) is slightly less efficient than DKLs24, our amortized efficiency—both in terms of communication and computation—is significantly better when $\ell = n/4$ or $\ell = n/3$, for both CL-MtA and OT-MtA instantiations. Notably, the amortized communication cost per signature per party becomes asymptotically constant. Readers are referred to Figures 8 and 9 for a detailed illustration[4].

In the honest-majority setting, we compare our robust scheme to the state-of-the-art robust protocols WMC24 [59] and TX25 [58]. When $\ell = 1$, both our offline and online communication costs are linear, making our scheme less efficient than WMC24 in that case. However, our amortized communication is better, particularly in the online phase. Furthermore, our online runtime is significantly faster than WMC24's, even without packing, and our offline runtime outperforms theirs when $\ell = n/3$. Compared with TX25, our amortized efficiency is significantly superior to it in both communication and computation. See Figure 10 and Table 4 for more details.

1.2. Technical Overviews

Recall the main challenge of threshold ECDSA is to compute k^{-1} and $k^{-1}x$ with each party holding additive shares (x_i, k_i). Most protocols, including the round-minimal DKLs24 [32] under dishonest-majority settings, use MtA to address this issue. We begin by reviewing the skeleton of MtA-based schemes.

Here, computing k^{-1} involves another secret nonce γ. The s-component can be rewritten as $s = \dfrac{\gamma(m + r \cdot x)}{\gamma k} \bmod q$. Assume n parties generate an ECDSA public key $X = xG$ and each one has a share x_i of secret key x.

- Round 1&2 (offline phase): Each party randomly selects k_i and γ_i. For each pair of (P_i, P_j), P_i takes k_i (resp. x_i) as input to run two-party MtA with each P_j, who takes γ_j as input. In total, $2n(n-1)$ MtA calls are needed to let each party get an additive share δ_i (resp. $\hat{\delta}_i$) of $k\gamma$ (resp. $x\gamma$). At the same time, kG can be computed to derive the component r.

4. To ensure fairness, the comparison is conducted under the same threshold value t, considering that our scheme requires the addition of $\ell - 1$ signers.

- Round 3 (online phase): Given a message $m \in \mathbb{Z}_q$, output δ_i and $\chi_i = m\gamma_i + r\hat{\delta}_i$. After receiving all $\{\delta_j, \chi_j\}_j$, the s component can be assembled as $\sum_j \chi_j / \sum_j \delta_j$.

Here, MtA costs dominate the total overhead. As each party must be involved in $O(n)$ MtAs, each one needs to send $O(n)$ transcripts and perform $O(n)$ computations.

Using packed secret sharing. In the above protocol, $\{k_i\}_i$ are shares of one secret value k. Thus, MtA operations in Round 1&2 will only yield one k-related result. Our intuitive idea to improve efficiency is that, if $\{k_i\}_i$ can represent the shares of multiple secret values, then the original MtA operations can produce multiple results. The idea is meaningful in this context because, with proper design tying and adjusting, the results corresponding to different k's can be different signatures.

The intuitive idea is realized using packed secret sharing [37], which is an extension of Shamir's secret sharing [55]. Recall that in Shamir's secret sharing, the secret value is placed in the constant term $f(0)$ of a secret t-degree polynomial $f(\cdot)$, and $f(i)$ is the share held by party P_i. In packed secret sharing, besides placing a secret value in $f(0)$, other secret values can be placed in $f(-1), f(-2), \cdots$, while keeping P_i's share as $f(i)$[5]. In this case, the polynomial degree needs to be increased for security. For ℓ-pack, we need $f(\cdot)$ to be $(t+\ell-1)$-degree, since now we need to protect ℓ values against t corrupted parties. Shamir Secret Sharing is actually the case $\ell = 1$. With no less than $t+\ell$ shares $f(i)$'s, ℓ secret values $f(0), f(-1), \cdots, f(-\ell+1)$ can be reconstructed by Lagrange interpolation.

If this technique can be applied to threshold ECDSA, it could improve the current setting, where each party requires $O(t+1)$ MtA calls to generate a single signature when the number of signatories is set to the minimum $n = t+1$. Specifically, it could be reduced to $O(t+\ell)$ MtA calls to generate ℓ signatures, resulting in an amortized overhead of $\frac{O(t+\ell)}{\ell}$ per signature. When t and ℓ are of the same order of magnitude, this leads to significant efficiency gains and reduced complexity.

Now, we give more details about such an optimization for threshold ECDSA. In the offline phase, we can pack the process of ℓ secret values $k^{(0)}, \cdots, k^{(\ell-1)}$ and $\gamma^{(0)}, \cdots, \gamma^{(\ell-1)}$, as well as the fixed signing key x, replicated ℓ times as a repeated secret. More precisely, the ℓ-packed secret sharing can build $f_{\mathbf{k}}(\cdot)$ s.t. $f_{\mathbf{k}}(-j) = k^{(j)}$ with $\{k_i = f_{\mathbf{k}}(i)\}_{i\in[n]}$ being shares. At the same time, signing key sharing should be accordingly upon $f_{\mathbf{x}}(\cdot)$ s.t. $f_{\mathbf{x}}(0) = \cdots = f_{\mathbf{x}}(-\ell+1) = x$. In this way, $O(t+\ell)$ MtA calls per party can handle the conversion of ℓ multiplications $k^{(j)}\gamma^{(j)}$'s to additions. The same holds for handling $x\gamma^{(j)}$'s. These results can be further used to generate ℓ signatures, which are produced independently one by one, rather than all at once. More specifically, in the online phase, given a hashed message m with index $0 \le j \le \ell-1$, the corresponding Lagrange coefficients are used to assemble $k^{(j)}\gamma^{(j)}$ and $(m + r^{(j)}x)\gamma^{(j)}$.

For non-robust schemes, $O(1)$ amortized communication can be achieved with the above techniques. However, robust schemes still face additional obstacles.

Amortizing online communication in a robust scheme. The $O(n)$ online communication overhead in previous robust threshold ECDSA schemes [58], [60] stems primarily from the fact that each party's signature share contains $O(n)$ MtA outputs that cannot be aggregated before being opened. As a result, these MtA outputs must be revealed individually. This is because the Lagrange coefficients, needed for aggregation, can only be calculated after identifying a sufficiently large subset of (semi-)honest parties to ensure robustness.

To leverage amortization, we reuse signature shares across ℓ signatures. To enable this, the ℓ messages and their corresponding public nonces must be embedded into the signature shares. Specifically, when computing the packed shares of $m+rx$ for ℓ different messages $(m^{(0)}, \ldots, m^{(\ell-1)})$ and corresponding r-components $(r^{(0)}, \ldots, r^{(\ell-1)})$, we interpolate them into public polynomials $f_{\mathbf{m}}(\cdot)$ and $f_{\mathbf{r}}(\cdot)$ of degree $\ell-1$, such that $f_{\mathbf{m}}(-j) = m^{(j)}$ and $f_{\mathbf{r}}(-j) = r^{(j)}$ for each j. Each party P_i then locally computes the interpolation value $f_{\mathbf{m}}(i) + f_{\mathbf{r}}(i)x_i$, which serves as a packed share corresponding to the ℓ values $\{m^{(j)} + r^{(j)}x\}_j$.

However, the degree of the above sharing polynomial $f_{\mathbf{m}} + f_{\mathbf{r}}f_{\mathbf{x}}$ is $(t+2\ell-2)$. This is a limitation that we want to avoid, as now we need at least $n \ge 2t+2\ell-1$ signatories, where $2t$ appears due to robustness. The packing parameter ℓ is decreased from $n-2t$ to $\lfloor \frac{n-2t+1}{2} \rfloor$, which means that performing $O(n)$ MtA operations per party will produce fewer signatures, reducing the overall amortization efficiency. Next, we give a method to remove this limitation.

Minimizing the polynomial degree. As discussed above, we aim to minimize the degree of the secret-sharing polynomials. In the case of ℓ-packed secret sharing, the minimum achievable degree is $t+\ell-1$.

To achieve this, we take the s-component as following

$$s = \frac{\gamma(m + r\cdot x)}{\gamma k} = r \cdot \frac{\gamma(mr^{-1} + x)}{\gamma k} = r \cdot \frac{\gamma(\hat{r} + x)}{\gamma k} \quad (1)$$

where $\hat{r} = mr^{-1}$. This transformation converts the scalar multiplication rx into an addition $\hat{r} + x$, enabling the replacement of polynomial multiplication with polynomial addition. In packed setting, the public $\hat{r}$'s can be interpolated into a polynomial $p(\cdot)$ with $p(-j) = \hat{r}^{(j)}$. Party P_i can locally compute $p(i) + x_i$ as its own share of $(\hat{r} + x)$'s ℓ different values. Currently, the degree of the sharing polynomial $p + f_{\mathbf{x}}$ remains $t+\ell-1$. Let $[\mathbf{r}]_d$ denote the degree d packed Shamir secret sharing of $\mathbf{r}$, and let $\circ$ denote the element-wise multiplication. Our approach essentially replaces the operation $[\mathbf{r}]_{\ell-1} \circ [\mathbf{x}]_{t+\ell-1} = [\mathbf{r} \circ \mathbf{x}]_{t+2\ell-2}$ by $[\hat{\mathbf{r}}]_{\ell-1} + [\mathbf{x}]_{t+\ell-1} = [\hat{\mathbf{r}} + \mathbf{x}]_{t+\ell-1}$. By combining the computed values of $\hat{r} + x$ and γx, ℓ different values of the numerator $\gamma(\hat{r} + x)$ in eq. (1) can be generated through appropriate linear combinations. In this way, we overcome the aforementioned limitation, allowing $O(n)$ MtA calls

5. There is a requirement that the set $\{0, -1, -2, \cdots, -(\ell-1)\}$ and parties' indices $\{1, 2, \cdots, n\}$ are disjoint.

to generate the maximum number of signatures. For more details, please refer to Figure 6.

The online amortizing in the robust scheme requires the online phase to be executed in batches, necessitating that ℓ messages be signed simultaneously. We discuss this limitation and its application scenarios in more detail in Section 1.3.

1.3. Discussions

Extension. Our scheme can be extended to support batch generation of signatures under multiple *distinct* public keys, meaning different secret keys are shared among the same set of n parties. Specifically, given ℓ distinct signing keys $\{x^{(j)}\}_j$, we can naturally define the sharing polynomial $f_{\mathbf{x}}(\cdot)$ such that $f_{\mathbf{x}}(-j) = x^{(j)}, \forall j \in [0, \ell-1]$, instead of using a single signing key x as before. In this way, the resulting ℓ signatures will each correspond to a different public key $X^{(j)} = x^{(j)}G$.

This extension suits the scenario where a cryptocurrency exchange (e.g., Binance) has multiple independent hot wallets corresponding to different public keys. When it sends bitcoins from multiple wallets, it needs to sign transactions using multiple signing keys. Given these signing keys shared (over $f_{\mathbf{x}}$) among n parties in a packed way, our extended scheme can sign these transactions with batch (pre-)processing.

Batched online phase of robust scheme. In our non-robust scheme, the online phase can sign each message independently. However, in the robust scheme, to exploit amortization and reduce the linear communication cost to a constant, the online phase must wait until ℓ messages are available for batch processing. A similar requirement arises in the threshold Schnorr scheme Sprint [6], which also employs packed secret sharing.

While this is a limitation, there are indeed scenarios where signers do not need to respond to signing requests immediately and can collect several requests and sign them in batches. For example, certificate authority (CA) servers periodically collect a set of incoming certificate requests and handle them in batch mode. This mode aligns well with operational considerations like rate limiting, load balancing, or scheduled downtime. The batched online phase is also well-suited for cryptocurrency exchanges. On receiving multiple withdrawal requests within a short period of time, the exchange processes them in batches. In this case, the above extension supporting multiple distinct public keys can be used to generate signatures simultaneously for the transactions sending bitcoins from multiple wallets.

Nevertheless, we note that the offline phase can always be executed in batches with constant communication overhead. We also present an alternative online phase that supports signing one by one, suffering linear communication. As analyzed in Remark 1 of Section 4, the linear component in communication is $64n$ bytes for 128-bit security.

We leave the design of a robust threshold ECDSA scheme that achieves constant online communication while supporting one-by-one signing as an open problem for future work.

Large party profile. The core idea conveyed in this paper is simple: while $t+1$ parties can only produce one signature at a time, $t+\ell$ parties can simultaneously generate ℓ signatures. As a result, by adding $\ell - 1$ additional signers, even as the number of parties n increases, the overall and per-party communication overhead can be amortized across ℓ signatures.

This approach becomes particularly beneficial in settings with a large number of parties (e.g., $n \geq 10$). While current commercial asset custody platforms tend to focus on small-scale deployments, NIST's call for multi-party threshold schemes envisions a broader range of scenarios—including medium ($9 \leq n \leq 64$), large ($65 \leq n \leq 1024$), and even enormous ($n \geq 1025$) party configurations [12, Table 3].

Packed secret sharing in general MPC. We observe that packed secret sharing has been used to amortize communication overhead in generic MPC protocols [34], [35], [42]. While threshold ECDSA can also be constructed from these generic MPCs, these constructions typically incur a higher round count and overall overhead [27]. This is because prior threshold ECDSA protocols have employed specialized techniques tailored to the task, rather than relying directly on generic MPC. Motivated by this, our work focuses on challenges specific to threshold ECDSA. In particular, we explore the use of packed secret sharing to reduce the per-party complexity of MtA calls and securely compute signatures based on their outputs. Additionally, the MPC approaches [34], [35], [42] need the addition of $2\ell - 2$ signers for the packing parameter ℓ. Our tailored design reduces this value to $\ell - 1$.

1.4. Related Works

Tang et al. [57] proposed a batch method for threshold ECDSA by aggregating range proofs across multiple MtA executions, though it does not reduce the overall complexity. Another batching technique appears in [48] under honest-majority settings, where pseudorandom secret sharing [25] is used to reduce communication costs.

For dishonest-majority threshold ECDSA, alternative techniques have been explored. Abram et al. [1] leveraged pseudorandom correlation generators (PCGs) [10], [11] to construct threshold ECDSA, achieving bandwidth savings of 1–2 orders of magnitude. However, their method requires a trusted setup and incurs high computational costs. Boneh et al. [7] proposed a universal thresholdizer based on threshold fully homomorphic encryption, but the resulting threshold ECDSA is limited by poor efficiency.

Under honest super-majority ($n \geq 3t + 1$) settings, several related works exist. Gennaro et al. [39] proposed a robust threshold ECDSA protocol, but it only tolerates halting adversaries and not malicious ones. In contrast, our work aims to achieve robustness against malicious parties. Groth and Shoup [43] introduced the first threshold ECDSA protocol assuming asynchronous networks, while most prior

works, including ours, focus on the setting of synchronous networks.

2. Preliminaries

Let $\mathbb{Z}_q$ be the finite field with q elements $\{0, 1, \cdots, q-1\}$ and $[a]$ 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 $\lambda \in \mathbb{N}$ denote the security parameter. Through this paper, n denotes the number of all participants, t denotes the threshold value, and ℓ denotes the packing parameter. For a row-vector $\mathbf{a} = [a_1, \cdots, a_k]$, $\mathbf{a}^\mathsf{T}$ is the transpose of $\mathbf{a}$, which is a column vector.

2.1. ECDSA

Let $pp = (\mathbb{G}, G, q)$ denote the group-generator-order tuple associated with the curve used in ECDSA signatures. The ECDSA scheme [26] utilizes a public algorithm $\mathsf{F} : \mathbb{G} \to \mathbb{Z}_q$ where $\mathsf{F}(R)$ outputs R's x-coordinate over $\mathbb{Z}_q$, and a hash function $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_q$. It operates as follows.

- $\mathsf{KGen}(pp, 1^\lambda) \to (x, X)$: output the (private) signing key $x \leftarrow\$ \mathbb{Z}_q$ and the public key $X = xG$.
- $\mathsf{Sign}(x, \mathsf{msg}) \to \sigma$: choose $k \leftarrow\$ \mathbb{Z}_q$ and compute $R = kG$, output $\sigma = (r, s)$ with $r = \mathsf{F}(R) \in \mathbb{Z}_q$ and $s = k^{-1} \cdot (m + rx) \bmod q$ with $m = \mathsf{H}(\mathsf{msg})$.
- $\mathsf{Verify}(X, (r, s), \mathsf{msg}) \to 0/1$: output 1 if $r = \mathsf{F}(R')$ where $R' = s^{-1}(\mathsf{H}(\mathsf{msg})G + rX)$.

2.2. Packed Secret Sharing

Let $\{P_i\}_{i\in[n]}$ be all parties and $\mathbf{x} = [x^{(0)}, \cdots, x^{(\ell-1)}] \in \mathbb{Z}_q^\ell$ be all secrets that are packed in one sharing. In this work, we use $[\mathbf{x}]_d$ to denote a degree-d packed sharing of $\mathbf{x} \in \mathbb{Z}_q^\ell$ with $d = t + \ell - 1$.

- $\mathsf{PackSS}(\mathbf{x}, t) \to [\mathbf{x}]_d$: A dealer picks a random degree-d polynomial $f(\cdot) \in \mathbb{Z}_q[X]$ such that $f(-j) = x^{(j)}, j \in [0, \ell-1]$, the i-th share held by party P_i is $x_i = f(i)$ for each $i \in [n]$.
- $\mathsf{Recover}(\{x_i\}_{i\in S}) \to \mathbf{x}/\perp$: If $s = |S| \geq d+1$ and $S = \{i_1, \cdots, i_s\} \subseteq [n]$, denote the Lagrange coefficient corresponding to the set S and point $-j$ as $\lambda_i^{(-j)} = \prod_{k\in S, k\neq i} \frac{-j-k}{i-k} \bmod q$ and return

$$\begin{pmatrix} x^{(0)} \\ x^{(1)} \\ \vdots \\ x^{(\ell-1)} \end{pmatrix} = \underbrace{\begin{pmatrix} \lambda_{i_1}^{(0)} & \lambda_{i_2}^{(0)} & \cdots & \lambda_{i_s}^{(0)} \\ \lambda_{i_1}^{(-1)} & \lambda_{i_2}^{(-1)} & \cdots & \lambda_{i_s}^{(-1)} \\ \vdots & \vdots & & \vdots \\ \lambda_{i_1}^{(-\ell+1)} & \lambda_{i_2}^{(-\ell+1)} & \cdots & \lambda_{i_s}^{(-\ell+1)} \end{pmatrix}}_{\boldsymbol{\Lambda}_S = [\lambda_i^{(-j)}]_{i\in S}^{j\in[0,\ell-1]} \in \mathbb{Z}_q^{\ell\times s}} \cdot \underbrace{\begin{pmatrix} x_{i_1} \\ x_{i_2} \\ x_{i_3} \\ \vdots \\ x_{i_s} \end{pmatrix}}_{\mathbf{s}}$$

Otherwise return $\perp$.

The correctness can be derived from the interpolation equation $x^{(j)} = f(-j) = \boldsymbol{\lambda}_S^{(-j)} \cdot \mathbf{s} \in \mathbb{Z}_q$ if $|S| \geq d+1$ where $\boldsymbol{\lambda}_S^{(-j)} = [\lambda_i^{(-j)}]_{i\in S}$ denotes the row-vector of $\boldsymbol{\Lambda}_S$ corresponding to $-j$. The t-privacy relies on the fact that any t shares from a random degree-d polynomial are independent of the ℓ secrets $\mathbf{x}$.

The packed secret sharing scheme is linear homomorphic: for all $\mathbf{x}, \mathbf{y} \in \mathbb{Z}_q^\ell$, $[\mathbf{x} + \mathbf{y}]_d = [\mathbf{x}]_{d_1} + [\mathbf{y}]_{d_2}$ with $d = \mathsf{max}(d_1, d_2)$. It also has the following multiplicative property: Let $\circ$ denote the coordinate-wise multiplication operation. For all $\mathbf{x}, \mathbf{y} \in \mathbb{Z}_q^\ell$ and $d_1 + d_2 < n$, $[\mathbf{x} \circ \mathbf{y}]_{d_1+d_2} = [\mathbf{x}]_{d_1} \circ [\mathbf{y}]_{d_2}$.

2.3. Packed Distributed Randomness Generation

In this work, we will use a packed distributed randomness generation (PDRG) protocol. It allows the parties to generate a packed secret sharing $[\mathbf{k}]_{t+\ell-1}$ for secrets $\mathbf{k} = [k^{(0)}, \cdots, k^{(\ell-1)}] \in \mathbb{Z}_q^\ell$, along with the corresponding public points $\{R^{(\nu)} = k^{(\nu)}G\}_\nu$, and reveals each party's public share $R_i = k_i G \in \mathbb{G}$.

In Figure 1, we define two functionalities for PDRG, namely the weak version [33] and the strong version [16].

- $\mathcal{F}_{\mathsf{wPDRG}}$: the ideal adversary Sim may abort the functionality. In its realization, if one party finds the received share is incorrect, it aborts the protocol.
- $\mathcal{F}_{\mathsf{sPDRG}}$: it can guarantee the delivery of $(R^{(0)}, \cdots, R^{(\ell-1)})$. In its realization, each honest party can publicly identify and exclude the cheaters' contributions and produce the correct output.

In the threshold ECDSA without robustness, the weak version $\mathcal{F}_{\mathsf{wPDRG}}$ is sufficient. The realization of $\mathcal{F}_{\mathsf{wPDRG}}$ is a two-round protocol, based on a packed variant of Pedersen's verifiable secret sharing (VSS) [53]. More precisely, each party (as a dealer) picks a random polynomial $f_i(\cdot)$ and runs Pedersen's VSS to share it with other parties. The correctness of each share can be verified by the recipient using Pedersen's commitments. If all shares are correct, the final combined sharing corresponds to $\sum_{i\in[n]} f_i(\cdot)$; otherwise, the protocol aborts. Please refer to Appendix C.1 for details.

However, $\mathcal{F}_{\mathsf{sPDRG}}$ is required for the robust threshold ECDSA. It will inform all parties about the identity of the parties who attempt to disrupt the protocol and will exclude them. Its realization is also a two-round protocol, based on publicly verifiable (packed) secret sharing (PVSS) [16]. Here, the "publicly verifiable" property ensures that the correctness of each share can be verified by all parties, not just the recipient. Thus, each honest party can consistently *identify* the misbehaving parties and *discard* their shares. As a result, the final combined sharing is computed over $\sum_{i\in S} f_i(\cdot)$, where S denotes the set of (semi-)honest parties. We adopt [16]'s PVSS construction over the class group. For details, please refer to Appendix C.2.

2.4. Multiplicative-to-Additive Protocol

The main building block of threshold ECDSA is a two-party MtA protocol. Specifically, we require a functionality that allows P_A and P_B, who hold values a and

Figure 1: Weak and Strong PDRG Functionalities

This functionality is parameterized by the party count n, the threshold t, and the packing parameter $\ell > 0$ such that $n \geq t+\ell$. The adversary Sim can corrupt up to t parties that are indexed by $\mathcal{P}^*$.

- Upon receiving $(\mathsf{randgen}, sid)$ from some party P_i such that sid is fresh, send $(\mathsf{randgen}, sid, i)$ to Sim.
- After receiving $(\mathsf{randgen}, sid)$ from all parties, if sid is agreed upon, then
  a) sample $k^{(0)}, k^{(1)}, \cdots, k^{(\ell-1)} \leftarrow\$ \mathbb{Z}_q$
  b) compute $R^{(0)} = k^{(0)}G, \cdots, R^{(\ell-1)} = k^{(\ell-1)}G$
  c) receive $(\mathsf{advshares}, sid, \{k_i\}_{i\in\mathcal{P}^*})$ from Sim
  d) sample a random polynomial $f(\cdot)$ of degree $t+\ell-1$ over $\mathbb{Z}_q$, subject to $f(i) = k_i$ for each $i \in \mathcal{P}^*$ and $f(-j) = k^{(j)}$ for each $j \in [0, \ell-1]$
  e) for $i \in [n]$, compute $k_i = f(i)$ and $R_i = k_iG$
  f) for $i \in [n]$, send (sid, k_i) to P_i
  g) receive (public, sid) from all parties
  h) for $i \in [n]$, send $(sid, (R^{(0)}, \cdots, R^{(\ell-1)}), (R_1, \cdots, R_n))$ to P_i.
- **Weak**: After receiving abort instruction from Sim during the above process, the functionality $\mathcal{F}_{\mathsf{wPDRG}}$ aborts.
- **Strong**: After receiving $(\mathsf{abort}, sid, j \in \mathcal{P}^*)$ instruction from Sim during the above process, $\mathcal{F}_{\mathsf{sPDRG}}$ records $C = C \cup \{j\}$ where C is a set of misbehaving parties. For each $i \in [n]$, $\mathcal{F}_{\mathsf{sPDRG}}$ sends $(sid, (R^{(0)}, \cdots, R^{(\ell-1)}), (R_i)_{i\in S})$ to P_i as the public output in step (h) with $S = [n] \setminus C$.

b respectively, to compute and learn α and β such that $\alpha + \beta = a \cdot b \bmod q$. This is also referred to as Oblivious Linear Evaluation (OLE) in some contexts [32], [41].

For this work, we define the basic functionality of MtA [33] and introduce a variant, MtA with public checking, as shown in Figure 2. MtA with public checking is specifically designed to identify cheaters publicly. If P_B's private input b is inconsistent with the revealed $B \in \mathbb{G}$, the functionality can flag P_B as malicious. This ensures that all honest parties can consistently exclude the contributions of malicious parties, which is essential for robust threshold ECDSA. Notably, the correctness of P_A's input a is not a concern in the application of threshold ECDSA (cf. Section 3.2).

There have been many techniques [14], [30], [32], [38], [49] in the literature of MPC for realizing the basic functionality $\mathcal{F}_{\mathsf{MtA}}$. Typically, one can use homomorphic encryption schemes (e.g., CL, Paillier, or JL) or OT protocols to realize it. Among them, CL-based MtA can achieve the best communicational efficiency, while OT-based one can give the best computational efficiency. As an instance, CL-based construction is provided in Appendix C.3. For the OT-based scheme, please refer to [30], [31], [32].

We present a realization of $\mathcal{F}_{\mathsf{MtAwPC}}$ based on CL homomorphic encryption. This construction augments the basic CL-based MtA with a zero-knowledge proof that ensures P_B uses the correct input b. The proof is publicly verifiable, allowing anyone to check the honesty of P_B. For the concrete construction, see Appendix C.3.

Figure 2: MtA (with Public Checking) Functionality

This functionality involves two parties, referred to as P_A and P_B, and is defined with respect to a prime q, which serves as the modulus for the multiplications performed.

- On receiving $(\mathsf{input}, sid, P_A||P_B, a)$ from P_A, if $a \in \mathbb{Z}_q$ and sid is fresh, then sample $\beta \leftarrow\$ \mathbb{Z}_q$, store (sid, a, β) in memory, and send $(\mathsf{output}, sid, P_A||P_B, \beta)$ to P_B.
- On receiving $(\mathsf{input}, sid_1||sid_2, P_A||P_B, a)$ from P_A, if $a \in \mathbb{Z}_q$ and sid_1, sid_2 are both fresh, then take it as two calls $(\mathsf{input}, sid_1, P_A||P_B, a)$ and $(\mathsf{input}, sid_2, P_A||P_B, a)$ from P_A.
- On receiving $(\mathsf{multiply}, sid, P_A||P_B, b)$ from P_B, if $b \in \mathbb{Z}_q$ and there exists a message of the form (sid, a, β) in memory, then compute $\alpha = a \cdot b - \beta \bmod q$, send $(\mathsf{output}, sid, P_A||P_B, \alpha)$ to P_A.
- **Public Checking**: Upon receiving $(\mathsf{check}, sid, P_B, B)$ from some party P_i where $B \in \mathbb{G}$ is P_B's public value, if $B \neq bG$, send $(\mathsf{cheater}, sid, P_B)$ to P_i for $i \in [n]$. Note that P_i can be anyone, not just P_A.

3. Packed Threshold ECDSA Scheme

3.1. Distributed Key Generation

Generating a secret sharing of ECDSA signing key is similar to realizing the functionality $\mathcal{F}_{\mathsf{wPDRG}}$, but with a distinction that the packed secrets correspond to the same value (the signing key x).

More precisely, the ideal functionality of distributed key generation (DKG) $\mathcal{F}_{\mathsf{DKG}}$ only differs from $\mathcal{F}_{\mathsf{wPDRG}}$ (Figure 1) in (a)-(d): it picks $x \leftarrow\$ \mathbb{Z}_q$ and packs ℓ identical values into the polynomial $f(\cdot)$ with $f(-j) = x,\ \forall j \in [0, \ell-1]$. Finally, $\mathcal{F}_{\mathsf{DKG}}$ will output $(sid, X, (X_1, \cdots, X_n))$, where X is the ECDSA public key and $X_i = x_iG$ is P_i's public key share with $x_i = f(i)$. Due to this minor modification, its realization closely resembles $\mathcal{F}_{\mathsf{wPDRG}}$'s realization. Please refer to Appendix C.1.

3.2. Distributed Signing

Let S be the set of signers' indices. Recall that the signing party count should satisfy $|S| \geq t + \ell$ with the packing parameter ℓ. When $\ell = 1$, it is our non-packed scheme. Recall that distributed signing is to jointly generate $(r = \mathsf{F}(kG), s = \dfrac{\gamma(m + xr)}{\gamma k})$.

Offline Phase. The packed offline phase makes the parties get ℓ random EC points $(R^{(0)}, \cdots, R^{(\ell-1)})$, and ℓ values $r^{(\nu)} = \mathsf{F}(R^{(\nu)})$ in parallel. In Figure 3, we present its concrete construction. Intuitively, its design consists of two modules.

Generation of packed secret sharing. It first invokes $\mathcal{F}_{\mathsf{wPDRG}}$ to generate a packed secret sharing of $\mathbf{k} = [k^{(0)}, \cdots, k^{(\ell-1)}]$ over polynomial $f_{\mathbf{k}}(\cdot)$. From this, each party P_i will hold a share $k_i = f_{\mathbf{k}}(i) \in \mathbb{Z}_q$. At the same time, the correct public share $R_i = k_iG$ is revealed and ℓ public EC points $[R^{(0)}, \cdots, R^{(\ell-1)}]$ are open.

Figure 3: Packed Offline Phase of Signing Protocol

Each party P_i holds the private input x_i and operates as follows.

Round 1. P_i picks $k_i, \gamma_i \leftarrow\!\!\$\ \mathbb{Z}_q$, computes $R_i = k_i G$, and sends

- $(\mathsf{input}, sid_{1,i,j} || sid_{2,i,j}, P_i || P_j, \gamma_i)$ to $\mathcal{F}_{\mathsf{MtA}}$ for each $j \neq i$
- $h_i = \mathsf{H}_0(R_i)$ to P_j for each $j \in S \setminus \{i\}$.

Round 2. On receiving

- $(\mathsf{output}, sid_{1,j,i}, P_j || P_i, \beta_{i,j})$ from $\mathcal{F}_{\mathsf{MtA}}$ for $j \in S \setminus \{i\}$
- $(\mathsf{output}, sid_{2,j,i}, P_j || P_i, \hat{\beta}_{i,j})$ from $\mathcal{F}_{\mathsf{MtA}}$ for $j \in S \setminus \{i\}$
- h_j for $j \in S \setminus \{i\}$,

P_i computes $\mathcal{B}_{i,j} = \beta_{i,j} G, \hat{\mathcal{B}}_{i,j} = \hat{\beta}_{i,j} G$ and sends

- $(\mathsf{multiply}, sid_{1,j,i}, P_j || P_i, k_i)$ to $\mathcal{F}_{\mathsf{MtA}}$ for each $j \in S \setminus \{i\}$
- $(\mathsf{multiply}, sid_{2,j,i}, P_j || P_i, x_i)$ to $\mathcal{F}_{\mathsf{MtA}}$ for each $j \in S \setminus \{i\}$
- $(\mathcal{B}_{i,j}, \hat{\mathcal{B}}_{i,j}, R_i)$ to P_j for each $j \in S \setminus \{i\}$

Output. On receiving

- $(\mathsf{output}, sid_{1,i,j}, P_i || P_j, \alpha_{i,j})$ from $\mathcal{F}_{\mathsf{MtA}}$ for $j \in S \setminus \{i\}$
- $(\mathsf{output}, sid_{2,i,j}, P_i || P_j, \hat{\alpha}_{i,j})$ from $\mathcal{F}_{\mathsf{MtA}}$ for $j \in S \setminus \{i\}$
- $(\mathcal{B}_{j,i}, \hat{\mathcal{B}}_{j,i}, R_j)$ from P_j for each $j \in S \setminus \{i\}$,

P_i checks $\mathcal{B}_{j,i} + \alpha_{i,j} G \stackrel{?}{=} \gamma_i R_j$, $\hat{\mathcal{B}}_{j,i} + \hat{\alpha}_{i,j} G \stackrel{?}{=} \gamma_i X_j$, and $h_j \stackrel{?}{=} \mathsf{H}_0(R_j)$ for each j. If it fails, P_i aborts.

Finally, P_i gets multiple R's: $R^{(\nu)} = \sum_{j \in S} \psi_j^{(\nu)} R_j$ for $\nu \in [\ell]$ and $\{\alpha_{i,j}, \hat{\alpha}_{i,j}, \beta_{i,j}, \hat{\beta}_{i,j}\}_{j \neq i}$.

For sharing temporary $\boldsymbol{\gamma} = [\gamma^{(0)}, \ldots, \gamma^{(\ell-1)}]$ that will be eliminated, each party randomly picks γ_i as its own packed share. It can be viewed as a packed secret sharing that does not involve a threshold structure. The assembled $\boldsymbol{\gamma} = \boldsymbol{\Lambda}_S [\gamma_i]_{i \in S}^{\mathsf{T}}$ is random in $\mathbb{Z}_q^\ell$, even if the t shares of γ_i are controlled by an adversary[6]. Therefore, these ℓ values of γ effectively mask k and x for security.

Splitting $(\gamma_i k_j, \gamma_i x_j)$ into additive shares. Each pair of parties P_i, P_j invoke $\mathcal{F}_{\mathsf{MtA}}$ for splitting $\gamma_i k_j = \alpha_{i,j} + \beta_{j,i}$[7]. By a MAC-style statistical check [32], P_i can check whether P_j contributes correct k_j via verifying whether $\gamma_i R_j = \alpha_{i,j} G + \mathcal{B}_{j,i}$ holds, vice versa. The case of splitting $\gamma_i x_j$ is similar.

Notably, there is no concern about ensuring the consistency of two MtA invocations for $\gamma_i k_j, \gamma_i x_j$. The two calls to $\mathcal{F}_{\mathsf{MtA}}$ with the same input γ_i are merged into one in the form of $(sid_{1,i,j} || sid_{2,i,j}, P_i || P_j, \gamma_i)$. In its realization, the first rounds of these two MtAs can also be merged into one, functioning as a two-dimensional vector OLE, thereby ensuring consistency naturally, like [14], [32], [38].

Online Phase. Although the generated offline outputs can help us to produce ℓ signatures, our online phase can sign the messages one by one without waiting for all ℓ messages to arrive. The online phase is shown in Figure 4.

We use the denominator $k^{(\nu)} \gamma^{(\nu)}$ of eq. (2) as a representative to demonstrate that our assembly method is correct. For each index $\nu \in [0, \ell - 1]$, we use the Lagrange coefficients $\{\lambda_i^{(-\nu)}\}_i$ to assemble these shares. To expand each secret value, we have

$$k^{(\nu)} \gamma^{(\nu)} = \left(\sum_i \lambda_i^{(-\nu)} \gamma_i\right) \left(\sum_j \lambda_j^{(-\nu)} k_j\right) = \sum_{i \in S} \lambda_i^{(-\nu)} a_i$$

since a_i (as in Figure 4) is reconstructed by aggregating all the shares of $\gamma_i k_j$ using the corresponding weights $\lambda_j^{(-\nu)}$.

6. The reason is that the matrix $\boldsymbol{\Lambda}_S$ is super-invertible [3], [44], meaning each ℓ-by-ℓ submatrix is invertible.

7. For a share $\alpha_{i,*}$ or $\beta_{i,*}$, the first index i denotes its owner.

Figure 4: Online Phase of Signing Protocol

On input of the ν-th message msg for each $\nu \in [0, \ell - 1]$, each party P_i holding (x_i, k_i, γ_i) and offline outputs does as follows.

One-Round Sending. P_i computes

- $m = \mathsf{H}(\mathsf{msg}), r = \mathsf{F}(R^{(\nu)})$
- $a_i = (\psi_i^{(\nu)})^2 \gamma_i k_i + \sum_{j \neq i} \psi_j^{(\nu)} \psi_j^{(\nu)} (\alpha_{i,j} + \beta_{i,j})$
- $b_i = \psi_i^{(\nu)} \lambda_i \gamma_i x_i + \sum_{j \neq i} (\psi_i^{(\nu)} \lambda_j \hat{\alpha}_{i,j} + \psi_j^{(\nu)} \lambda_i \hat{\beta}_{i,j})$
- $c_i = \psi_i^{(\nu)} m \gamma_i + r b_i$

and sends (a_i, c_i) to the aggregator.

Output. On receiving (a_j, c_j) from each P_i, $i \in S$, the aggregator generates

$$s = \frac{\sum_{i \in S} c_i}{\sum_{i \in S} a_i} \mod q \tag{2}$$

outputs the signature $sig = (r, s)$ if $\mathsf{Verify}(X, sig, \mathsf{msg}) \to 1$.

Asymptotic Complexity. We divide the signing protocol into offline and online phases. In the offline phase, each party invokes one time of $\mathcal{F}_{\mathsf{wPDRG}}$ and $O(n)$ times of $\mathcal{F}_{\mathsf{MtA}}$. According to their realizations (Appendices C.1 and C.3), the sending communication complexity per party is $O(n)$. The computation complexity per party is $O(n^2)$, due to the EC group operations. With $\ell = \epsilon n$ signatures generated from offline outputs, the amortized communication and computation cost per party per signature are $O(1)$ and $O(n)$, respectively. The online phase only costs $O(1)$ communication and $O(n)$ field operations per party to generate a signature.

4. Packed Threshold ECDSA with Robustness

In this part, we add robustness to our above scheme under honest-majority settings ($n \geq 2t + \ell$). Recall that for generating the maximum number of signatures in parallel (cf. Section 1.2), we rewrite the s-component

$$s = r \cdot \frac{\gamma(mr^{-1} + x)}{\gamma k}. \tag{3}$$

DKG. To guarantee DKG's output, we need a functionality similar to $\mathcal{F}_{\mathsf{sPDRG}}$, with the difference being that the sharing polynomial $f_{\mathbf{x}}(\cdot)$ packs ℓ identical values. We call it as $\mathcal{F}_{\mathsf{sDKG}}$. Due to space limitations, we defer its realization to Appendix C.2.

Offline signing phase. For adding robustness to distributed signing, the calls to $\mathcal{F}_{\mathsf{wPDRG}}$ and $\mathcal{F}_{\mathsf{MtA}}$ should be replaced by $\mathcal{F}_{\mathsf{sPDRG}}$ and $\mathcal{F}_{\mathsf{MtAwPC}}$, respectively. This idea is similar to that of the previous robust work [58]. In order to ensure the correctness of honest parties' signature shares, all honest

Figure 5: Packed Offline Phase of Robust Signing Protocol.

Each party P_i holds the private input x_i and operates as follows.

Round 1 and *Round 2* follow the same structure as in Figure 3, but with $\mathcal{F}_{\mathsf{wPDRG}}$ and $\mathcal{F}_{\mathsf{MtA}}$ replaced by $\mathcal{F}_{\mathsf{sPDRG}}$ and $\mathcal{F}_{\mathsf{MtAwPC}}$, respectively.

Output. On receiving

- $(sid_0, (R^{(0)}, \cdots, R^{(\ell-1)}), (R_i)_{i\in S})$ from $\mathcal{F}_{\mathsf{sPDRG}}$, where S denotes the set of parties that did not send an abort instruction to $\mathcal{F}_{\mathsf{sPDRG}}$
- $(\mathsf{output}, sid_{1,i,j}, P_i||P_j, \alpha_{i,j})$ from $\mathcal{F}_{\mathsf{MtAwPC}}$ for $j \in S \setminus \{i\}$
- $(\mathsf{output}, sid_{2,i,j}, P_i||P_j, \hat{\alpha}_{i,j})$ from $\mathcal{F}_{\mathsf{MtAwPC}}$ for $j \in S \setminus \{i\}$

Public Checking: P_i sends $(\mathsf{check}, sid_{1,\mu,j}, P_j, R_j)$ and $(\mathsf{check}, sid_{2,\mu,j}, P_j, X_j)$ to $\mathcal{F}_{\mathsf{MtAwPC}}$ for each $j \in S \setminus \{i\}$ and $\mu \in S \setminus \{j\}$. If receiving $(\mathsf{cheater}, sid_{1,\mu,j}, P_j)$ or $(\mathsf{cheater}, sid_{2,\mu,j}, P_j)$ for some μ, then set $S = S \setminus \{j\}$.

If $|S| \geq t + \ell$, P_i gets the offline outputs.

- $r^{(0)} = \mathsf{F}(R^{(0)}), \cdots, r^{(\ell-1)} = \mathsf{F}(R^{(\ell-1)})$
- $\delta_{i,j} = \alpha_{i,j} + \beta_{i,j}, \hat{\delta}_{i,j} = \hat{\alpha}_{i,j} + \hat{\beta}_{i,j}$ for each $j \in S \setminus \{i\}$

parties need to find cheaters and exclude their contributions before generating signature shares. Otherwise, even a sufficient number of honest parties would not guarantee a successful signing execution. The previously invoked $\mathcal{F}_{\mathsf{wPDRG}}$ and $\mathcal{F}_{\mathsf{MtA}}$ cannot realize it, but $\mathcal{F}_{\mathsf{sPDRG}}$ and $\mathcal{F}_{\mathsf{MtAwPC}}$ can, without adding the communication rounds and while preserving constant amortized communication overhead. The offline phase is shown in Figure 5.

Online signing phase. Another obstacle occurs in the online phase. Since the subset of (semi-)honest parties can only be determined in the final stage, a_i, b_i (in Figure 4) cannot be aggregated beforehand. As a result, each one sends individual $\{a_{i,j}, b_{i,j}\}_j$, which leads to linear communication overhead, as in previous MtA-based robust works [58], [60].

In this work, fortunately, this cost can be amortized since these shares are used to produce ℓ signatures in batch. In Figure 6, we present the concrete construction.

Specifically, we use a polynomial $p(\cdot)$ to represent ℓ values mr^{-1}'s, which requires ℓ messages to be provided as input simultaneously. The correctness is clear via the equation for each $\nu \in [0, \ell-1]$ with $p(-\nu) = m^{(\nu)}(r^{(\nu)})^{-1}$,

$$\begin{aligned}\gamma^{(\nu)}(p(-\nu)+x) &= \left(\sum_i \lambda_i^{(-\nu)}\gamma_i\right)\left(\sum_j \lambda_j^{(-\nu)}(p(j)+x_j)\right)\\ &= \sum_{i,j}\lambda_i^{(-\nu)}\lambda_j^{(-\nu)}b_{i,j}\end{aligned}$$

where each $b_{i,j}$ is masked via a zero's share $\hat{z}(j)$ (as in Figure 6). The aggregator randomly selects a party subset $\bar{S}$ of size $t+\ell$. If the produced signatures based on $\bar{S}$ are valid, then output them. Otherwise, re-select a new subset $\bar{S}$ and reassemble again. Since there must be $t+\ell$ honest parties, the subset producing valid signatures must exist.

We have another method to achieve robustness without selecting the subset $\bar{S}$, following [58]. It is to add a cheater identification mechanism. Specifically, each party, when revealing its signature share, also generates a proof that the share was correctly produced. This enables the aggregator to naturally identify $t+\ell$ (semi-)honest parties. However, for each signature, the aggregator must perform $O(n^2)$ EC group operations to filter out malicious outputs. In contrast, our online protocol described above only requires finite field operations, which are almost free compared with EC group operations. Therefore, as long as the number of combinations $\binom{|S|}{t+\ell}$ is not excessively large, we still opt for the approach of subset selection.

Figure 6: Packed Online Phase of Robust Signing Protocol

On input of ℓ messages $(\mathsf{msg}^{(0)}, \cdots, \mathsf{msg}^{(\ell-1)})$ and the offline outputs, each party P_i holding (x_i, k_i, γ_i) does as follows. Denote the set S as the current party set, which is updated after excluding the misbehaving ones in the offline phase.

One-Round Sending. P_i computes

- $m^{(0)} = \mathsf{H}(\mathsf{msg}^{(0)}), \cdots, m^{(\ell-1)} = \mathsf{H}(\mathsf{msg}^{(\ell-1)})$
- the fixed degree-$(\ell-1)$ polynomial $p(\cdot)$ with $p(-\nu) = m^{(\nu)}(r^{(\nu)})^{-1}$ for each $\nu \in [0, \ell-1]$
- two random degree-$(t+\ell-1)$ polynomials $z(\cdot), \hat{z}(\cdot)$ with $z(-\nu) = \hat{z}(-\nu) = 0$ for each $\nu \in [0, \ell-1]$.
- $a_{i,i} = \gamma_i k_i + z(i) \bmod q$, $\{a_{i,j} = \delta_{i,j} + z(j) \bmod q\}_{j\in S\setminus\{i\}}$
- $b_{i,i} = \gamma_i(p(i)+x_i) + \hat{z}(i) \bmod q$, $\{b_{i,j} = \gamma_i p(j) + \hat{\delta}_{i,j} + \hat{z}(j) \bmod q\}_{j\in S\setminus\{i\}}$

and sends $\{a_{i,j}, b_{i,j}\}_{j\in S}$ to the aggregator.

Output. On receiving $\{a_{i,j}, b_{i,j}\}_{j\in S}$ from P_i, $i \in S$, the aggregator does the following

1. pick a subset $\bar{S} = \{i_1, i_2, \cdots, i_{t+\ell}\} \subseteq S$: set the matrices

$$\underbrace{\begin{bmatrix} a_{i_1,i_1} & \cdots & a_{i_{t+\ell},i_1} \\ \vdots & \ddots & \vdots \\ a_{i_1,i_{t+\ell}} & \cdots & a_{i_{t+\ell},i_{t+\ell}} \end{bmatrix}}_{\boldsymbol{A}_{\bar{S}}}, \quad \underbrace{\begin{bmatrix} b_{i_1,i_1} & \cdots & b_{i_{t+\ell},i_1} \\ \vdots & \ddots & \vdots \\ b_{i_1,i_{t+\ell}} & \cdots & b_{i_{t+\ell},i_{t+\ell}} \end{bmatrix}}_{\boldsymbol{B}_{\bar{S}}}.$$

2. For $\mathsf{msg}^{(\nu)}, \forall\ \nu \in [0, \ell-1]$, it generates[a]

$$s^{(\nu)} = r^{(\nu)} \cdot \frac{\boldsymbol{\lambda}_{\bar{S}}^{(-\nu)} \boldsymbol{B}_{\bar{S}} (\boldsymbol{\lambda}_{\bar{S}}^{(-\nu)})^{\mathsf{T}}}{\boldsymbol{\lambda}_{\bar{S}}^{(-\nu)} \boldsymbol{A}_{\bar{S}} (\boldsymbol{\lambda}_{\bar{S}}^{(-\nu)})^{\mathsf{T}}} \bmod q, \tag{4}$$

   restart from Step 1 if $\mathsf{Verify}(X, (r^{(\nu)}, s^{(\nu)}), \mathsf{msg}^{(\nu)}) \to 0$.
3. Output $\mathsf{sig}^{(\nu)} = (r^{(\nu)}, s^{(\nu)}),\ \forall\ \nu \in [0, \ell-1]$.

a. $\boldsymbol{\lambda}_{\bar{S}}^{(-\nu)} = [\lambda_{i_1}^{(-\nu)}, \lambda_{i_2}^{(-\nu)}, \cdots, \lambda_{i_{t+\ell}}^{(-\nu)}]$

Remark 1 (Signing one by one). As discussed in 1.3, the above online phase requires waiting for ℓ messages as input. It can also be adjusted to sign messages one by one, without waiting for other messages.

On input of the ν-th hashed message $m = m^{(\nu)}$ and ν-th nonce $r = r^{(\nu)}$ *only*, the non-packed online phase involves the following changes.

1. The polynomials $z(\cdot), \hat{z}(\cdot)$ are fresh for each message, and are now zero *only* at the point $-\nu$.
2. The generation of $\{a_{i,j}\}_j$ remains unchanged, but $\{b_{i,j}\}_j$ are changed: $b_{i,i} = \gamma_i(mr^{-1} + x_i) + \hat{z}(i), \{b_{i,j} =$

$\gamma_i mr^{-1} + \hat{\delta}_{i,j} + \hat{z}(j)\}_j$.

Using the coefficients $\boldsymbol{\lambda}_{\bar{S}}^{(-\nu)}$ can produce $m^{(\nu)}$'s signature. For security reasons, using the Lagrange coefficients corresponding to other interpolation points will yield a random value that conceals secret values, due to the Change 1.

However, this prevents amortization in the online phase, resulting in $O(n)$ communication sent per signature per party. On the other hand, the linear portion actually grows slowly with only $64n$ bytes for a 128-bit security level.

Asymptotic Complexity. In this robust scheme, each party invokes one time of $\mathcal{F}_{\mathsf{sPDRG}}$ and $O(n)$ times of $\mathcal{F}_{\mathsf{MtAwPC}}$. According to their realizations (Appendices C.2 and C.3), each party sends $O(n)$ transcripts and performs $O(n^2)$ computations. With the packing parameter $\ell = n - 2t$, the amortized complexity per signature is $O(1)$ and $O(n)$. It is important to note that during the online phase, assembling each signature requires $O(n^2)$ field operations. Since it is almost free compared with other operations, we excluded it from asymptotic complexity analysis.

5. Security Analysis

We present a threshold ECDSA functionality in Figure 7 that our non-robust scheme realizes, which is borrowed from [32], [33]. Unlike prior works, we divide the signing process into an offline phase that generates multiple R's in batch, and an online phase that consumes them one by one.

Theorem 1. *On the common input $(\mathbb{G}, G, q, n, t, \ell)$ with $n \geq t + \ell$, our non-robust scheme in Section 3 UC-realizes the threshold ECDSA functionality $\mathcal{F}_{\mathsf{ECDSA}}$ in the $(\mathcal{F}_{\mathsf{DKG}}, \mathcal{F}_{\mathsf{wPDRG}}, \mathcal{F}_{\mathsf{MtA}})$-hybrid model against a malicious adversary that statically corrupts up to t fixed parties.*

Proof. The theorem is equivalent to the following statement. For every malicious adversary $\mathcal{A}$ that statically corrupts up to t parties, there exists a simulator $\mathsf{Sim}^{\mathcal{A}}$ 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

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

where Π denotes our scheme. We begin by constructing the simulator $\mathsf{Sim}^{\mathcal{A}}$ which is parameterized by $(\mathbb{G}, G, q, n, t, \ell)$, then we will analyze that it produces a view for the environment that is indistinguishable from the real world.

The simulator has oracle access to the adversary $\mathcal{A}$ and simulates for it the executions of our protocols. It forwards all messages from its own environment $\mathcal{Z}$ to $\mathcal{A}$, and vice versa. When the simulation begins, $\mathcal{A}$ announces the identities of up to t corrupted parties. Let the indices of these parties be given by $\mathcal{P}^* \subset S$. In the simulated execution, $\mathsf{Sim}^{\mathcal{A}}$ will interact with $\mathcal{A}$ on behalf of every honest party and on behalf of the ideal oracles $\mathcal{F}_{\mathsf{DKG}}, \mathcal{F}_{\mathsf{wPDRG}}, \mathcal{F}_{\mathsf{MtA}}$. At the same time, the simulator (as an ideal adversary) interacts with the ideal functionality $\mathcal{F}_{\mathsf{ECDSA}}$ on behalf of every corrupt party.

Figure 7: Threshold ECDSA Functionality $\mathcal{F}_{\mathsf{ECDSA}}$

This functionality is parameterized by the EC group parameters $(\mathbb{G}, G, q)$, the party count n, the threshold t, and the packing parameter ℓ s.t. $n \geq t + \ell$. The ideal adversary Sim can corrupt up to t parties that are indexed by $\mathcal{P}^*$. During the key generation phase, Sim may instruct the functionality to abort. During any signing phase, Sim may instruct the functionality to fail, but in this case, the functionality does not halt, and further signing queries may be attempted. This functionality ensures synchrony using the framework of Katz et al. [47], meaning it only advances from one round to the next when all honest parties indicate that it should.

Key Generation.

- On receiving (keygen, sid) from some party P_i for $i \in [n]$ where sid is fresh, send $(\mathsf{keygen}, sid, i)$ to Sim.
- After receiving (keygen, sid) from all parties $P_1, \cdots, P_n$, if sid is agreed-upon, then
  1. sample $x \leftarrow\$ \mathbb{Z}_q$, compute $X = xG$ and store (sid, x, X)
  2. send X as the public key to Sim
  3. if receiving (abort, sid) from Sim, send (abort, sid) to all parties; otherwise send X as the public key to all parties.
  4. ignore any further call to keygen.

Offline Signing.

- On receiving $(\mathsf{offsign}, sid, \ell)$ from some party P_i, if $\ell \in O(1)$, keygen was already called and $sid = (\cdots, S)$ is fresh with $|S| = t + \ell$, send $(\mathsf{offsign}, sid, \ell, i)$ to Sim.
- After receiving $(\mathsf{offsign}, sid, \ell)$ from all parties in set S, sample $k^{(\nu)} \leftarrow\$ \mathbb{Z}_q$ and compute $R^{(\nu)} = k^{(\nu)}G$ for each $\nu \in [0, \ell - 1]$.
- Send $\{R^{(\nu)}\}_\nu$ to Sim. If receiving (fail, sid) from Sim, send (fail, sid) to all parties and store (failed, sid) in memory. Otherwise, send $\{R^{(\nu)}\}_\nu$ to all parties and store $(\mathsf{offsigned}, sid)$ in memory.

Online Signing.

- On receiving $(\mathsf{sign}, sid||\nu, m)$ from some party P_i, if $(\mathsf{offsigned}, sid)$ exists in memory, $sid||\nu$ is fresh and $\nu \in [0, \ell - 1]$, send $(\mathsf{sign}, sid||\nu, m, i)$ to Sim.
- After receiving $(\mathsf{sign}, sid||\nu, m)$ from all parties in S, retrieve $k = k^{(\nu)}$ and compute $r = \mathsf{F}(R^{(\nu)})$, $s = \dfrac{m + rx}{k} \bmod q$.
- Send (r, s) to Sim. If receiving $(\mathsf{fail}, sid||\nu)$ from Sim, send $(\mathsf{fail}, sid||\nu)$ to all parties and store $(\mathsf{failed}, sid||\nu)$ in memory. Otherwise, send (r, s) to all parties and store $(\mathsf{signed}, sid||\nu)$ in memory.

Key Generation.

1. On receiving $(\mathsf{keygen}, sid, i)$ for some $i \in [n] \setminus \mathcal{P}^*$ from $\mathcal{F}_{\mathsf{ECDSA}}$, reply (keygen, sid) on behalf of $P_j, j \in \mathcal{P}^*$
2. On receiving X as the public key from $\mathcal{F}_{\mathsf{ECDSA}}$, Sim simulates $\mathcal{F}_{\mathsf{DKG}}$ for $\mathcal{A}$.
   a) On receiving $(\mathsf{randgen}, sid)$ from each corrupted party $P_j, j \in \mathcal{P}^*$ on behalf of $\mathcal{F}_{\mathsf{DKG}}$, send $(\mathsf{randgen}, sid, i)$ for $i \in [n] \setminus \mathcal{P}^*$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{DKG}}$.
   b) On receiving $(\mathsf{advshares}, sid, \{x_j\}_{j\in\mathcal{P}^*})$ from $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{DKG}}$, build a degree-$(t + \ell - 1)$ polynomial $f_{\mathbf{X}}(\cdot)$ over $\mathbb{G}$ s.t.

- $f_{\mathbf{X}}(-\nu) = X$ for each $\nu \in [0, \ell-1]$ and
- $f_{\mathbf{X}}(j) = x_j G$ for each $j \in \mathcal{P}^*$.

c) For each $i \in [n]$, set $X_i = f_{\mathbf{X}}(i) \in \mathbb{G}$ and send $(sid, X, (X_1, \cdots, X_n))$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{DKG}}$.

3. If receiving abort instruction from $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{DKG}}$, then send (abort, sid) to $\mathcal{F}_{\mathsf{ECDSA}}$.

Offline Signing.

1. On receiving $(\mathsf{offsign}, sid, \ell, i)$ for some $i \in S \setminus \mathcal{P}^*$ from $\mathcal{F}_{\mathsf{ECDSA}}$, reply $(\mathsf{sign}, sid, \ell)$ on behalf of $P_j, j \in \mathcal{P}^*$.
2. On receiving $\{R^{(\nu)}\}_{\nu \in [0,\ell-1]}$ from $\mathcal{F}_{\mathsf{ECDSA}}$, Sim simulates *the offline phase of threshold signing*.
   a) On receiving $(\mathsf{randgen}, sid_0)$ from each corrupted party $P_j, j \in \mathcal{P}^*$ on behalf of $\mathcal{F}_{\mathsf{wPDRG}}$, send $(\mathsf{randgen}, sid_0, i)$ for $i \in S \setminus \mathcal{P}^*$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{wPDRG}}$.
   b) On receiving $(\mathsf{advshares}, sid_0, \{k_j\}_{j \in \mathcal{P}^*})$ from $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{wPDRG}}$, build a degree-$(t+\ell-1)$ polynomial $f_{\mathbf{R}}(\cdot)$ over $\mathbb{G}$ s.t.
   - $f_{\mathbf{R}}(-\nu) = R^{(\nu)}$ for each $\nu \in [0, \ell-1]$ and
   - $f_{\mathbf{R}}(j) = k_j G$ for each $j \in \mathcal{P}^*$.

   c) For each $i \in S$, set $R_i = f_{\mathbf{R}}(i) \in \mathbb{G}$ and send $(sid_0, \{R^{(0)}, \cdots, R^{(\ell-1)}\}, (R_i)_i)$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{wPDRG}}$.
   d) On receiving $(\mathsf{input}, sid_{1,j,i} || sid_{2,j,i}, P_j || P_i, \gamma_j)$ from each corrupted party $P_j, j \in \mathcal{P}^*$ on behalf of $\mathcal{F}_{\mathsf{MtA}}$, pick $\alpha_{j,i}, \hat{\alpha}_{j,i} \leftarrow\!\!\$\, \mathbb{Z}_q$ for each $i \in S \setminus \mathcal{P}^*$,
   - reply $(\mathsf{output}, sid_{1,j,i}, P_j || P_i, \alpha_{j,i})$ to P_j
   - reply $(\mathsf{output}, sid_{2,j,i}, P_j || P_i, \hat{\alpha}_{j,i})$ to P_j
   - send $\mathcal{B}_{i,j} = \gamma_j R_i - \alpha_{j,i} G$, $\hat{\mathcal{B}}_{i,j} = \gamma_j X_i - \hat{\alpha}_{j,i} G$ to P_j.

   e) On behalf of $\mathcal{F}_{\mathsf{MtA}}$, for each $i \in S \setminus \mathcal{P}^*, j \in \mathcal{P}^*$,
   - pick $\beta_{j,i}, \hat{\beta}_{j,i} \leftarrow\!\!\$\, \mathbb{Z}_q$
   - send $(\mathsf{output}, sid_{1,i,j}, P_i || P_j, \beta_{j,i})$ to P_j
   - send $(\mathsf{output}, sid_{2,i,j}, P_i || P_j, \hat{\beta}_{j,i})$ to P_j

   f) Compute $\delta_{j,i} = \alpha_{j,i} + \beta_{j,i}$ and $\hat{\delta}_{j,i} = \hat{\alpha}_{j,i} + \hat{\beta}_{j,i}$.
   g) After receiving $(\mathsf{multiply}, sid_{1,i,j}, P_i || P_j, k'_j)$ and $(\mathsf{multiply}, sid_{2,i,j}, P_i || P_j, x'_j)$ on behalf of $\mathcal{F}_{\mathsf{MtA}}$, and $\mathcal{B}_{j,i}, \hat{\mathcal{B}}_{j,i}$ from P_j, if $\exists\ j \in \mathcal{P}^*, i \in S \setminus \mathcal{P}^*$, $k_j \neq k'_j$, $x_j \neq x'_j$ or $\mathcal{B}_{j,i} \neq \beta_{j,i} G$, $\hat{\mathcal{B}}_{j,i} \neq \hat{\beta}_{j,i} G$, or $\mathcal{A}$ sends abort instruction to $\mathcal{F}_{\mathsf{wPDRG}}$, then send (fail, sid) to $\mathcal{F}_{\mathsf{ECDSA}}$.

Online Signing.

1. We assume that no fail message was sent to $\mathcal{F}_{\mathsf{ECDSA}}$.
2. Pick $\bar{x} \leftarrow\!\!\$\, \mathbb{Z}_q$ and regard $\bar{x}$ as the dummy secret key of ECDSA.
3. With the knowledge of $\{x_j\}_{j \in \mathcal{P}^*}$, build a $(t+\ell-1)$-degree polynomial $f_{\bar{\mathbf{x}}}(\cdot)$ over $\mathbb{Z}_q$ s.t.
   - $f_{\bar{\mathbf{x}}}(0) = f_{\bar{\mathbf{x}}}(-1) = \cdots = f_{\bar{\mathbf{x}}}(-\ell+1) = \bar{x}$ and
   - $f_{\bar{\mathbf{x}}}(j) = x_j$ for each $j \in \mathcal{P}^*$.

4. On receiving $(\mathsf{sign}, sid || \nu, m, i)$ from some $i \in S \setminus \mathcal{P}^*$, reply $(\mathsf{sign}, sid || \nu, m)$ on behalf of $P_j, j \in \mathcal{P}^*$.
5. On receiving (r, s) from $\mathcal{F}_{\mathsf{ECDSA}}$ where $r = \mathsf{F}(R^{(\nu)})$, Sim simulates *the online phase of threshold signing*.
   a) Compute $\bar{k} = \dfrac{m + r\bar{x}}{s}$ and regard $\bar{k}$ as the dummy secret nonce used for the ν-th signature.
   b) With the knowledge of $\{k_j\}_{j \in \mathcal{P}^*}$, build a degree-$(t+\ell-1)$ polynomial $f_{\bar{\mathbf{k}}}(\cdot)$ over $\mathbb{Z}_q$ s.t. $f_{\bar{\mathbf{k}}}(-\nu) = \bar{k}$ and $f_{\bar{\mathbf{k}}}(j) = k_j$ for each $j \in \mathcal{P}^*$.
   c) For each honest party $P_i, i \in S \setminus \mathcal{P}^*$, simulate shares $\bar{x}_i = f_{\bar{\mathbf{x}}}(i), \bar{k}_i = f_{\bar{\mathbf{k}}}(i)$.
   d) With the knowledge of $\gamma_j, \delta_{j,i}, \hat{\delta}_{j,i}$, simulate honest parties' shares for each $j \in \mathcal{P}^*, i \in S \setminus \mathcal{P}^*$: pick $\gamma_i \leftarrow\!\!\$\, \mathbb{Z}_q$ and compute $\delta_{i,j} = \gamma_j \bar{k}_i + \gamma_i k_j - \delta_{j,i}$, $\hat{\delta}_{i,j} = \gamma_j \bar{x}_i + \gamma_i x_j - \hat{\delta}_{j,i}$.
   e) For each honest party P_i, $i \in S \setminus \mathcal{P}^*$, using $(\gamma_i, \bar{x}_i, \bar{k}_i, \{\delta_{i,j}, \hat{\delta}_{i,j}\}_{j \in \mathcal{P}^*})$ to simulate:
   - for $j \in S \setminus \mathcal{P}^*$, $a_{i,j} = \gamma_i \bar{k}_j$, $b_{i,j} = \gamma_i (m + r\bar{x}_j)$
   - for $j \in \mathcal{P}^*$, $a_{i,j} = \delta_{i,j}$, $b_{i,j} = m\gamma_i + r\hat{\delta}_{i,j}$

   send $a_i = \sum_{j \in S} \lambda_j^{(-\nu)} a_{i,j}$ and $b_i = \sum_{j \in S} \lambda_j^{(-\nu)} b_{i,j}$ to $\mathcal{A}$ on behalf of each honest party P_i.
   f) After receiving (a_j, b_j) for each $j \in \mathcal{P}^*$ from $\mathcal{A}$, if the assembled value s^* via eq. (2) is not not equal to s, then send $(\mathsf{fail}, sid || \nu)$ to $\mathcal{F}_{\mathsf{ECDSA}}$.

Analysis. We use a sequence of hybrid experiments to show the indistinguishability between ideal and real worlds. It begins with the real world, thus we have $\mathcal{H}_0 = \{\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda, z)\}_{\lambda \in \mathbb{N}, z \in \{0,1\}^*}$.

Hybrid $\mathcal{H}_1$. In this hybrid experiment, all honest parties and ideal functionalities $(\mathcal{F}_{\mathsf{DKG}}, \mathcal{F}_{\mathsf{wPDRG}}, \mathcal{F}_{\mathsf{MtA}})$ are replaced with Sim, who 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 experiment replaces the real execution of the distributed key generation protocol with the simulated execution as above. We show that it is indistinguishable.

Sim gets X as public key from $\mathcal{F}_{\mathsf{ECDSA}}$. It aims to simulate the functionality $\mathcal{F}_{\mathsf{DKG}}$ to output X if there is no abort instruction from the adversary. The polynomial $f_{\mathbf{X}}(\cdot)$ over $\mathbb{G}$ is built by Sim according to X and $\mathcal{A}$'s shares. The discrete logarithm polynomial $f_{\mathbf{x}}(\cdot)$ of $f_{\mathbf{X}}(\cdot)$ would be distributed identically to the one chosen in $\mathcal{F}_{\mathsf{DKG}}$. Thus, this simulation is perfect. We have $\mathcal{H}_2 = \mathcal{H}_1$.

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 statistically indistinguishable.

Sim gets $(r^{(0)}, \cdots, r^{(\ell-1)})$ from $\mathcal{F}_{\mathsf{ECDSA}}$. Then, from those, ℓ EC points $(R^{(0)}, \cdots, R^{(\ell-1)})$ can be easily derived. Firstly, it aims to simulate the functionality $\mathcal{F}_{\mathsf{wPDRG}}$ to output $(R^{(0)}, \cdots, R^{(\ell-1)})$ if there is no abort instruction from the adversary. This simulation is perfect.

At the same time, Sim also needs to simulate the functionality $\mathcal{F}_{\mathsf{MtA}}$. It is very easy via picking $\alpha_{j,i}, \hat{\alpha}_{j,i}, \beta_{j,i}, \hat{\beta}_{j,i} \leftarrow\!\!\$\, \mathbb{Z}_q$ for the adversary. The difference is that Sim computes $\mathcal{B}_{i,j} = \gamma_j R_i - \alpha_{j,i} G$ and $\hat{\mathcal{B}}_{i,j} = \gamma_j X_i - \hat{\alpha}_{j,i} G$, instead of $\mathcal{B}_{i,j} = \beta_{i,j} G, \hat{\mathcal{B}}_{i,j} = \hat{\beta}_{i,j} G$, since currently it does not know correct $\beta_{i,j}, \hat{\beta}_{i,j}$. Fortunately, simulating them in this manner will pass the corrupted party P_j's check.

On the other hand, after receiving P_j's $\mathcal{B}_{j,i}, \hat{\mathcal{B}}_{j,i}$, Sim does not check $\mathcal{B}_{j,i}+\alpha_{i,j}G \stackrel{?}{=} \gamma_i R_j$, $\hat{\mathcal{B}}_{j,i}+\hat{\alpha}_{i,j}G \stackrel{?}{=} \gamma_i X_j$ since it does not know correct $\alpha_{i,j}, \hat{\alpha}_{i,j}$ currently. Instead, it checks

$$\mathcal{B}_{j,i} \stackrel{?}{=} \beta_{j,i}G,\ \hat{\mathcal{B}}_{j,i} \stackrel{?}{=} \hat{\beta}_{j,i}G. \tag{5}$$

If the checking of (5) passes and Sim received $(\mathsf{multiply}, -, P_i||P_j, k_j)$, $(\mathsf{multiply}, -, P_i||P_j, x_j)$ on behalf of $\mathcal{F}_{\mathsf{MtA}}$, then Sim should be convinced that P_j performed honestly, which is the same as the real protocol. In a real protocol, if an adversary sends wrong input $k'_j \neq k_j$ or $x'_j \neq x_j$ to $\mathcal{F}_{\mathsf{MtA}}$, it has a $1/q$ probability of sending correct $\mathcal{B}_{j,i}$ or $\hat{\mathcal{B}}_{j,i}$ to pass the consistency checking. The probability that at least one of $\mathsf{poly}(\lambda)$ attempts will pass is upper bounded by $\mathsf{poly}(\lambda)/q$. It follows that $\mathcal{H}_3 \approx_s \mathcal{H}_2$.

Hybrid $\mathcal{H}_4$. This hybrid behaves identically to $\mathcal{H}_3$, except that the online signing protocol is replaced with a simulated execution. We show that the simulated transcript is distributed identically to the real one.

In $\mathcal{H}_4$, Sim utilizes (r, s) received from $\mathcal{F}_{\mathsf{ECDSA}}$ to simulate honest parties' signature shares. It picks a random $\bar{x} \leftarrow\!\!\$\ \mathbb{Z}_q$ and regards it as a dummy secret key. Under this dummy secret key, the corresponding dummy secret nonce can be derived with $\bar{k} = \frac{m+r\bar{x}}{s}$. Then, Sim can simulate honest parties' shares $\{\bar{x}_i, \bar{k}_i\}_{i\in S\setminus\mathcal{P}^*}$ of packed secret sharing, which can recover $\bar{\mathbf{x}} = (\bar{x}, \cdots, \bar{x})$ and $\bar{k}$ at the $-\nu$ point together with corrupted parties' shares. With these shares, Sim can invoke $\mathcal{A}$'s secret values $\delta_{j,i}, \hat{\delta}_{j,i}$ (on behalf of $\mathcal{F}_{\mathsf{MtA}}$) to get honest parties' output

$$\delta_{i,j} = \gamma_j \bar{k}_i + \gamma_i k_j - \delta_{j,i},\ \hat{\delta}_{i,j} = \gamma_j \bar{x}_i + \gamma_i x_j - \hat{\delta}_{j,i}.$$

Finally, Sim is able to generate the honest parties' signature shares (a_i, b_i) using dummy secret key share $\bar{x}_i$, dummy secret nonce share $\bar{k}_i$, randomly chosen $\gamma_i \leftarrow\!\!\$\ \mathbb{Z}_q$, and simulated MtA outputs $\{\delta_{i,j}, \hat{\delta}_{i,j}\}_{j\in\mathcal{P}^*}$. The simulated values have the same distribution as those in the real protocol due to the presence of random γ_i in both $\mathcal{H}_4$ and the real protocol. If $\mathcal{A}$ sends correct (a_j, b_j) on behalf of the corrupted parties for $j \in \mathcal{P}^*$, then eq. (2) will make $\frac{\sum_{i\in S}\lambda_i^{(-\nu)}b_i}{\sum_{i\in S}\lambda_i^{(-\nu)}a_i} = \frac{m+r\bar{x}}{k} = s$. If $\mathcal{A}$ cheats on their signature shares, the offset caused on s will be the same as that in the real protocol. It follows that $\mathcal{H}_4 = \mathcal{H}_3$.

It has $\mathcal{H}_4 = \left\{\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{ECDSA}},\mathsf{Sim}^{\mathcal{A}},\mathcal{Z}}(\lambda, z)\right\}_{\lambda\in\mathbb{N}, z\in\{0,1\}^*}$. By transitivity, we also have $\mathcal{H}_4 \approx_s \mathcal{H}_0$ where statistical difference between $\mathcal{H}_4$ and $\mathcal{H}_0$ is upper bounded by $\mathsf{poly}(\lambda)/q$. □

Our robust scheme realizes a robust variant of $\mathcal{F}_{\mathsf{ECDSA}}$, denoted as $\mathcal{F}_{\mathsf{rECDSA}}$. It will not abort or fail when receiving abort or fail instruction from Sim when $n \geq 2t + \ell$. At the same time, the functionality receives ℓ hashed messages $(m^{(0)}, \cdots, m^{(\nu-1)})$ and signs them in batches for the online signing phase.

Theorem 2. *On the common input $(\mathbb{G}, G, q, n, t, \ell)$ with $n \geq 2t+\ell$, our robust scheme in Section 4 UC-realizes $\mathcal{F}_{\mathsf{rECDSA}}$ in the $(\mathcal{F}_{\mathsf{sDKG}}, \mathcal{F}_{\mathsf{sPDRG}}, \mathcal{F}_{\mathsf{MtAwPC}})$-hybrid model against a malicious adversary that statically corrupts up to t fixed parties.*

This proof closely follows the proof of Theorem 1, with the difference being the simulation of the online signing phase. In this case, the ideal adversary Sim receives ℓ signatures at once and simulates the honest parties' signature shares accordingly to reconstruct them. The high-level idea for this simulation remains similar to that of the above proof. We omit the details due to space constraints.

6. Performance

Our implementations include (robust) threshold ECDSAs with CL-based MtA and OT-based MtA. The two approaches represent the two mainstream MtA constructions currently. CL is a linearly homomorphic encryption scheme [21], and the MtA based on it has high communication efficiency but low computational efficiency[8]. In contrast, the OT-based MtA has high computational efficiency but poor communication efficiency.

We provide benchmarking results for DKLs24 [32] and our non-robust scheme under dishonest-majority settings, as well as for WMC24 [59], TX25 [58] and our robust scheme under honest-majority settings. Notably, we do not include benchmarking results for KU24 [48] as it requires honest-majority assumptions without robustness, which doesn't align with the cases we focus on. Since these protocols have the same communication rounds (except for WMC24, which has one more round), we evaluate the protocols on a laptop and do not take network latency into account. Since our packed schemes require $\ell - 1$ additional signers, we ensure *fairness* by making comparisons under the same threshold value t. We always first determine t and ℓ, and then set the number of signers as $n = t + \ell$, or $n = 2t + \ell$ for robust schemes.

Testbed Environment. For the implementation based on CL encryption, we use an open-source C++ library `BICYCL`[9] [9]. For OT-based implementation, we use an open-source Rust library of DKLs24[10] [32]. Our implementation is available at anonymous github[11]. Experiments were run on a MacBook Air with macOS Sonoma Version 14.4, 8GB RAM, and an Apple M2 chip. The computational security level is set to be $\lambda = 128$ bits, and the statistical security parameter is $\lambda_s = 40$ bits. We use SHA256 to instantiate hash functions. ECDSA was taken over the elliptic curve (EC) secp256k1 with the group order of the 256-bit prime q.

With CL-based MtA. As shown in Table 2 and Figure 8, we compare our CL-based work with the CL-variant of

8. CL can be replaced by Paillier [52] and JL [45] homomorphic encryption schemes, but the modified schemes will be less efficient.

9. https://gite.lirmm.fr/crypto/bicycl

10. https://github.com/silence-laboratories/silent-shard-dkls23-ll

11. https://anonymous.4open.science/r/Packed-SSS-for-Threshold-ECDSA

TABLE 2: **The (amortized) costs of threshold ECDSAs with CL-MtA under dishonest-majority settings.** We set the number of signatories as $n = t + \ell$ for our schemes and the values represent the *amortized* costs, while CL-DKLs24 uses $n = t+1$. "CL-DKLs24" is a variant of DKLs24 replacing the original OT-based MtA with CL-based MtA. "Communication" represents each party's sending communication. "Time" represents each party's local runtime.

t	Communication (KB)						Time (ms)					
	4	8	12	16	20	24	4	8	12	16	20	24
CL-DKLs24 [32]	5.1	9.36	13.65	17.94	22.23	26.52	758	1494	2232	2974	3815	4488
$\ell = 1$ - Ours	6.09	11.25	16.42	21.58	26.75	31.9	768	1514	2262	3039	3865	4576
$\ell = \lfloor t/3 \rfloor$ - Ours	6.09	6.27	5.07	5.35	5.53	5.12	768	777	816	827	868	880
$\ell = \lfloor t/2 \rfloor$ - Ours	3.69	3.78	3.81	3.83	3.84	3.84	487	525	548	573	577	584

TABLE 3: **The (amortized) costs of threshold ECDSAs with OT-MtA under dishonest-majority settings.**

t	Communication (KB)						Time (ms)					
	4	8	12	16	20	24	4	8	12	16	20	24
DKLs24 [32]	212	424	636	848	1060	1272	28	56	92	115	141	169
$\ell = 1$ - Ours	213	426	639	852	1065	1278	30	63	101	139	180	219
$\ell = \lfloor t/3 \rfloor$ - Ours	213	240	200	213	222	206	30	38	38	44	53	60
$\ell = \lfloor t/2 \rfloor$ - Ours	133	146	151	153	154	155	20	26	32	40	48	59

TABLE 4: **The (amortized) costs of *robust* threshold ECDSAs under honest-majority settings.** We set $n = 2t + 1$ for WMC24 and TX25, and $n = 2t + \ell$ for our cases.

t	Offline-Comm. (KB)				Online-Comm. (KB)				Offline-Time (ms)				Online-Time (ms)			
	2	4	8	12	2	4	8	12	2	4	8	12	2	4	8	12
WMC24 [59]	4.1	4.1	4.1	4.1	0.8	0.8	0.8	0.8	1056	1794	3306	4790	553	891	1564	2235
TX25 [58]	6.12	11.3	22	32	0.36	0.6	1.11	1.6	1580	4085	12151	24289	1.51	3.68	9.9	18.7
$\ell = 1$ - Ours	6.8	12.2	22.99	33.78	0.31	0.56	1.06	1.56	1738	4494	13366	26718	0.31	1.18	2.19	4.58
$\ell = \lfloor t/2 \rfloor$ - Ours	6.8	6.77	6.76	6.75	0.31	0.31	0.31	0.31	1738	2679	4462	6223	0.31	0.56	1.16	2.67
$\ell = t - 1$ - Ours	6.8	4.9	4.4	4.3	0.31	0.2	0.2	0.2	1738	2097	3280	4519	0.31	0.38	1.14	2.23

Figure 8: **The trend line as t increases with CL-based MtA.** The y-axis is set to a *logarithmic* scale. "Ours-1" is our $\ell = 1$ case, "Ours-2" is our $\ell = \lfloor t/3 \rfloor$ (i.e., $\ell = n/4$) case, "Ours-3" is our $\ell = \lfloor t/2 \rfloor$ (i.e., $\ell = n/3$) case.

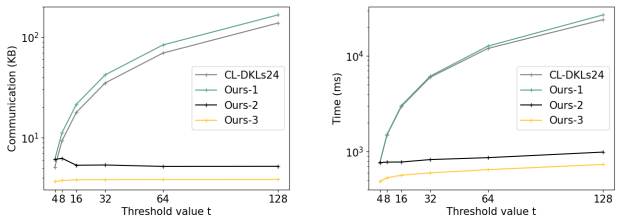
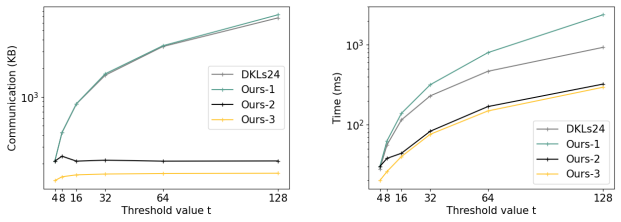


Figure 9: **The trend line as t increases with OT-based MtA.** The y-axis is set to a *logarithmic* scale. "Ours-1" is our $\ell = 1$ case, "Ours-2" is our $\ell = \lfloor t/3 \rfloor$ (i.e., $\ell = n/4$) case, "Ours-3" is our $\ell = \lfloor t/2 \rfloor$ (i.e., $\ell = n/3$) case.



DKLs24 [32], where the original OT-based MtA is replaced by CL-based MtA (denoted as CL-DKLs24). Specifically, the implementations of our scheme contain three settings:

- the packing parameter $\ell = 1$ (i.e., no packing);
- $\ell = \lfloor t/3 \rfloor$ (i.e., $\epsilon = 1/4$ and $\ell \approx n/4$);
- $\ell = \lfloor t/2 \rfloor$ (i.e., $\epsilon = 1/3$ and $\ell \approx n/3$).

All three settings can accommodate dishonest-majority settings.

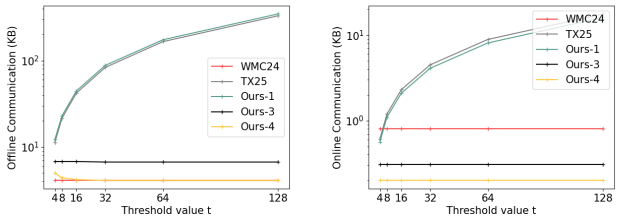
Compared with CL-DKLs24, our non-packed scheme incurs 20% more communication, while our running time is very close to theirs. On the other hand, our packed schemes with $\ell = \lfloor t/3 \rfloor$ and $\ell = \lfloor t/2 \rfloor$ are clearly superior to the CL-DKLs24 in both communication and computation.

With OT-based MtA As shown in Table 3 and Figure 9, we compare our OT-based construction with DKLs24. Our three settings of ℓ and n values are the same as those of CL-MtA schemes.

For our $\ell = 1$ case, our communication cost is close to DKLs24, but our running time is 30% higher than theirs. Fortunately, our packed OT-based scheme can amortize the communication into a constant value asymptotically. Moreover, our amortized running time is significantly lower than that of DKLs24, and it grows more slowly than theirs.

Robust Threshold ECDSAs. In honest-majority settings, the comparison of our robust scheme, WMC24 [59] and

Figure 10: **The trend line of offline and online communication for robust schemes as t increases.** The y-axis is set to a *logarithmic* scale. "Ours-1" is our $\ell = 1$ case, "Ours-3" is our $\ell = \lfloor t/2 \rfloor$ case (here, $\ell = n/5$), "Ours-4" is our $\ell = t - 1$ ($\ell = n/3$) case.



TX25 [58], is given in Table 4 and Figure 10. We implement our scheme in the settings of $\ell = 1$, $\ell = \lfloor t/2 \rfloor$ (i.e., $\ell \approx n/5$), and $\ell = t - 1$ (i.e., $\ell \approx n/3$) with the signing party count $n = 2t + \ell$.

WMC24 has constant communication, while our non-packed case requires linear communication in both the offline and online phases. It is advantageous that our packed cases can amortize it to a constant. In the offline phase, our constant is close to theirs when $\ell = t - 1$, and in the online phase, it is smaller when $\ell = \lfloor t/2 \rfloor$. In terms of runtime, our online is significantly faster than theirs, even without packing. However, our offline efficiency only surpasses theirs when $\ell = t - 1$, as a result of amortization.

The overhead of our $\ell = 1$ case is comparable to that of TX25. Since our online phase does not include a cheater identification mechanism, its overhead is slightly lower than that of TX25. Moreover, our packed schemes significantly outperform TX25 in both communication and computation costs.

References

[1] 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.

[2] Amit Agarwal, Alexander Bienstock, Ivan Damgård, and Daniel Escudero. Honest majority GOD MPC with O(depth(C)) rounds and low online communication. In *ASIACRYPT*, pages 234–265. Springer, 2025.

[3] Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure MPC with linear communication complexity. In *TCC*, volume 4948, pages 213–230. Springer, 2008.

[4] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *CCS*, pages 390–399. ACM, 2006.

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

[6] Fabrice Benhamouda, Shai Halevi, Hugo Krawczyk, Yiping Ma, and Tal Rabin. SPRINT: high-throughput robust distributed schnorr signatures. In *EUROCRYPT*, volume 14655, pages 62–91. Springer, 2024.

[7] 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.

[8] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *EUROCRYPT*, volume 9666, pages 327–357. Springer, 2016.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] Ignacio Cascudo and Bernardo David. SCRAPE: scalable randomness attested by public entities. In *ACNS*, volume 10355, pages 537–556. Springer, 2017.

[16] 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.

[17] Ignacio Cascudo, Bernardo David, Lydia Garms, and Anders Konring. YOLO YOSO: fast and simple encryption and secret sharing in the YOSO model. In *ASIACRYPT*, volume 13791, pages 651–680. Springer, 2022.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

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

[24] Ran Cohen and Yehuda Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. *Journal of Cryptology*, 30(4):1157–1186, 2017.

[25] Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *TCC*, volume 3378, pages 342–362. Springer, 2005.

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

[27] Anders P. K. Dalskov, Claudio Orlandi, Marcel Keller, Kris Shrishak, and Haya Schulmann. Securing DNSSEC keys via threshold ECDSA from generic MPC. In *ESORICS*, volume 12309, pages 654–673. Springer, 2020.

[28] Ivan Damgård, Thomas P. Jakobsen, Jesper Buus Nielsen, Jakob Illeborg Pagter, and Michael Bæksvang Østergaard. Fast threshold ECDSA with honest majority. *J. Comput. Secur.*, 30(1):167–196, 2022.

[29] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *CRYPTO*, volume 4622, pages 572–590. Springer, 2007.

[30] 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.

[31] 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.

[32] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Threshold ECDSA in three rounds. In *S&P*, pages 3053–3071. IEEE, 2024.

[33] 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.

[34] Daniel Escudero, Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Turbopack: Honest majority MPC with constant online communication. In *CCS*, pages 951–964. ACM, 2022.

[35] Daniel Escudero, Vipul Goyal, Antigoni Polychroniadou, Yifan Song, and Chenkai Weng. Superpack: Dishonest majority MPC with constant online communication. In *EUROCRYPT*, volume 14005, pages 220–250. Springer, 2023.

[36] 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.

[37] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *STOC*, pages 699–710. ACM, 1992.

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

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

[40] Rosario Gennaro, Darren Leigh, Ravi Sundaram, and William S. Yerazunis. Batching schnorr identification scheme with applications to privacy-preserving authorization and low-bandwidth communication devices. In *ASIACRYPT*, volume 3329, pages 276–292, 2004.

[41] Satrajit Ghosh, Jesper Buus Nielsen, and Tobias Nilges. Maliciously secure oblivious linear function evaluation with constant overhead. In *ASIACRYPT*, volume 10624, pages 629–659. Springer, 2017.

[42] Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Sharing transformation and dishonest majority MPC with packed secret sharing. In *CRYPTO*, volume 13510, pages 3–32. Springer.

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

[44] Martin Hirt and Jesper Buus Nielsen. Robust multiparty computation with linear communication complexity. In *CRYPTO*, volume 4117, pages 463–482. Springer, 2006.

[45] Marc Joye and Benoît Libert. Efficient cryptosystems from 2^{k}-th power residue symbols. *EUROCRYPT*, pages 76–92, 2013.

[46] 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.

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

[48] Jonathan Katz and Antoine Urban. Honest-majority threshold ecdsa with batch generation of key-independent presignatures. *IACR Cryptol. ePrint Arch.*, 2024.

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

[50] Yehuda Lindell. How to simulate it - A tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer International Publishing, 2017.

[51] 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.

[52] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, volume 1592, pages 223–238. Springer, 1999.

[53] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, volume 576, pages 129–140. Springer, 1991.

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

[55] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

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

[57] Guofeng Tang, Shuai Han, Li Lin, Changzheng Wei, and Ying Yan. Batch range proof: How to make threshold ECDSA more efficient. In *CCS*, pages 4256–4270. ACM, 2024.

[58] 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.

[59] 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.

[60] 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.

[61] 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.

[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. Castagnos-Laguillaumie Encryption

CL encryption scheme [21] works in an unknown-order class group $\hat{\mathfrak{G}}$ of order $q\hat{s}$, where the upper bound $\tilde{s}$ of $\hat{s}$ is known but $\hat{s}$ is unknown. One can find a subgroup $\mathfrak{F}$ generated by $\mathfrak{f} \in \hat{\mathfrak{G}}$ of order q. On input $\mathfrak{f}^x \in \mathfrak{F}$, there exists a *deterministic polynomial-time* algorithm $\mathsf{Dlog}(\cdot)$ to compute x. Let $\mathfrak{G} = \mathfrak{F} \times \mathfrak{G}^q$ is a cyclic subgroup of $\hat{\mathfrak{G}}$ where $\mathfrak{G}^q$ contains the q-th powers of elements in $\mathfrak{G}$. Note that $\mathfrak{G}$ is generated by $\mathfrak{g} = \mathfrak{f}\mathfrak{g}_q$ where $\mathfrak{g}_q$ is a generator of $\mathfrak{G}^q$. The decryption key and randomness used for encryption are sampled uniformly over the range $\mathcal{D}_q = [0, 2^{\lambda_s}\tilde{s}]$, such

that the distribution of $\{\mathfrak{g}_q^x : x \leftarrow\$ \mathcal{D}_q\}$ is at distance less than $2^{-\lambda_s}$ from the uniform distribution in $\mathfrak{G}^q$, where λ_s is the statistical parameter. The CL scheme and linear homomorphic operations are described below.

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

We describe the assumptions of the class group we will need for this work.

Definition 1 (HSM assumption [9], [22])**.** *For any probabilistic polynomial time (PPT) adversary $\mathcal{A}$, we have the advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{HSM}}$ is negligible in λ. Let $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{HSM}}$ be*

$$\left| \Pr\left[b = b^* \;\middle|\; \begin{array}{c} x \leftarrow\$ \mathcal{D}_q, u \leftarrow\$ \mathbb{Z}_q, \\ b \leftarrow\$ \{0,1\}, z_0 = \mathfrak{f}^u \mathfrak{g}_q^x, z_1 = \mathfrak{g}_q^x, \\ b^* \leftarrow \mathcal{A}(z_b) \end{array} \right] - \frac{1}{2} \right|.$$

Definition 2 (DDH-$\mathfrak{f}$ assumption [16], [22])**.** *For any PPT adversary $\mathcal{A}$, with $\mathcal{D} = [0, 2^{\lambda_s} \tilde{s} q]$, we have the advantage $\mathsf{Adv}_{\mathcal{A}}^{DDH\text{-}\mathfrak{f}}$ is negligible in λ. Let $\mathsf{Adv}_{\mathcal{A}}^{DDH\text{-}\mathfrak{f}}$ be*

$$|\Pr[b^* = b \,|x, y \leftarrow\$ \mathcal{D}, u \leftarrow\$ \mathbb{Z}_q, X = \mathfrak{g}^x, Y = \mathfrak{g}^y, b \in \{0,1\}, Z_0 = \mathfrak{g}^{xy}, Z_1 = \mathfrak{g}^{xy}\mathfrak{f}^u, b^* \leftarrow \mathcal{A}(X, Y, Z_b)] - 1/2|.$$

We first define a probabilistic algorithm $\hat{\mathfrak{G}} \leftarrow \mathsf{CLGen}(1^\lambda, q; r)$ where $r \in \{0,1\}^\lambda$ is the randomness used by CLGen and $\hat{\mathfrak{G}}$ is the class group we introduced above.

Definition 3 (Rough Order assumption ROC [13], [16])**.** *For a natural number $C \in \mathbb{N}$ and security parameter λ, consider $\mathcal{D}_C^{\mathsf{rough}}$ the uniform distribution in the set $\{r \in \{0,1\}^\lambda : \hat{\mathfrak{G}} \leftarrow \mathsf{CLGen}(1^\lambda, q; r) \wedge \forall$ prime $p < C, p \nmid \mathsf{ord}(\hat{\mathfrak{G}})\}$.*

$$\mathsf{Adv}_{\mathcal{A}}^{ROC} = |\Pr[b^* = b \,|r_0 \leftarrow\$ \{0,1\}^\lambda, r_1 \leftarrow\$ \mathcal{D}_C^{\mathsf{rough}}, b \leftarrow\$ \{0,1\}, b^* \leftarrow \mathcal{A}(1^\lambda, r_b)] - 1/2|$$

is negligible in λ for all PPT $\mathcal{A}$.

Appendix B. Zero-Knowledge Proofs

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 $\mathcal{L} = \{x | \exists\ w,$ s.t. $(x, w) \in \mathcal{R}\}$ as the set of instances x that have a witness w in the relation $\mathcal{R}$. In this work, we use a non-interactive zero-knowledge (NIZK) proof, which is always generated via the Fiat-Shamir transform [36] in the random oracle model [5] from an interactive ZK proof. To simplify the presentation, we denote NIZK for relation $\mathcal{R}$ as two algorithms $(\mathsf{Prove}_{\mathcal{R}}, \mathsf{Verify}_{\mathcal{R}})$:

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

The (NI)ZK proof system must be equipped with the following properties.

- Completeness: For any $(x, w) \in \mathcal{R}$ and $\pi_{\mathcal{R}} \leftarrow \mathsf{Prove}_{\mathcal{R}}(x, w)$, $\mathsf{Verify}_{\mathcal{R}}(x, \pi_{\mathcal{R}})$ always outputs 1.
- Zero-Knowledge: There exists a PPT simulator Sim that takes the statement $x \in \mathcal{L}$ as input and can output an accepting proof $\pi_{\mathcal{R}}$ which is statistically indistinguishable from a real one generated via $\mathsf{Prove}_{\mathcal{R}}(x, w)$.
- Standard Soundness: For every $x \notin \mathcal{L}$, no PPT prover can output a valid proof $\pi_{\mathcal{R}}$ s.t. $\mathsf{Verify}_{\mathcal{R}}(x, \pi_{\mathcal{R}}) \rightarrow 1$ with non-negligible probability.

Ideal Functionality. 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.

- Proof of Knowledge: There exists a PPT extractor $\mathsf{Ext}^{\mathsf{Prove}}$ that uses $\mathsf{Prove}(\cdot)$ as a black box and can extract a witness w' for $(x, w') \in \mathcal{R}$.

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 parties $P_1, \cdots P_n$ as follows:
> Upon receiving $(\mathsf{prove}, \mathsf{sid}, x, w, j)$ from party P_i, $\mathcal{F}_{\mathsf{zk}_\mathcal{R}}$ sends $(\mathsf{proved}, \mathsf{sid}, x, i)$ to P_j if $(x, w) \in \mathcal{R}$. Otherwise, it ignores the message. Note that if no j is specified, then $\mathcal{F}_{\mathsf{zk}_\mathcal{R}}$ sends $(\mathsf{proved}, \cdots)$ to all parties.

Relations. We use NIZK proofs for the following relations.

1. *Discrete Logarithm.* $\mathcal{R}_{\mathsf{DL}} = \{((A, B) \in \mathbb{G}, b \in \mathbb{Z}_q) : B = bA\}$. Its proof is well-studied [54].
2. *Knowledge of Multiple Pedersen Commitments.* Given the Pedersen parameter $(G, H) \in \mathbb{G}^2$, $\mathcal{R}_{\mathsf{Ped}} = \{(\{\mathcal{P}_i\}_{i\in[m]}, \{a_i, a_i'\}_{i\in[m]}) : \{\mathcal{P}_i = a_i G + a_i' H\}_{i\in[m]}\}$. This proof is a batched Schnorr-type zero-knowledge proof [40], which is a proof of knowledge according to the general forking lemma [4], [8].
3. *Knowledge of CL secret key.* $\mathcal{R}_{\mathsf{key}} = \{(ek, dk \in \mathcal{D}_q) : ek = \mathfrak{g}_q^{dk}\}$. We borrow this proof from [62, Algorithm 2].
4. *Well-formedness of a CL Ciphertext.* Given ek as CL's encryption key, define $\mathcal{R}_{\mathsf{enc}} = \{(\mathcal{C}, (m \in \mathbb{Z}_q, \rho \in \mathcal{D}_q)) : \mathcal{C} = \mathsf{CL.Enc}(ek, m, \rho)\}$. Its proof can be found in [13].
5. *CL Affine Operation.* Given $\mathcal{C}$ as a CL ciphertext under public key ek, define $\mathcal{R}_{\mathsf{aff}} = \{((\mathcal{C}', \mathcal{C}), (b, \beta \in \mathbb{Z}_q, \rho \in \mathcal{D}_q)) : \mathcal{C}' = b \odot \mathcal{C} \oplus \mathsf{CL.Enc}(ek, \beta, \rho)\}$. Please refer to [58] for this proof.
6. *CL Affine Operation and Discrete Logarithm.* Given $\mathcal{C}$ as a CL ciphertext under public key ek, define $\mathcal{R}_{\mathsf{aff\text{-}DL}} = \{((\mathcal{C}', \mathcal{C}, B), (b, \beta \in \mathbb{Z}_q, \rho \in \mathcal{D}_q)) : B = bG, \mathcal{C}' = b \odot \mathcal{C} \oplus \mathsf{CL.Enc}(ek, \beta, \rho)\}$. This proof is an extension of that of $\mathcal{R}_{\mathsf{aff}}$ with involving EC group operations.

7. *Validity of Encrypted Shares.* Given the set of indices S, degree d, and $\{ek_j\}_{j\in S}$ as a set of CL's encryption keys, define $\mathcal{R}_{\mathsf{Sh}} = \{(\{\mathcal{C}_{f(j)}\}_{j\in S}, (f(\cdot), \rho \in \mathcal{D}_q)) : \mathsf{deg}(f(\cdot)) \leq d, \mathcal{C}_{f(j)} = \mathsf{CL.Enc}(ek_j, f(j), \rho)\}$. For this relation, we refer the readers to [16, Figure 4].
8. *Discrete Logarithm and CL Decryption.* $\mathcal{R}_{\mathsf{DL\text{-}D}} = \{((\mathcal{C}_a, A), (a, dk)) : A = aG, a = \mathsf{CL.Dec}(dk, \mathcal{C}_a)\}$. Following [58], this proof is an extension of that for $\mathcal{R}_{\mathsf{enc}}$.
9. *Pedersen Commitments and CL Encryptions.* Given $\{ek_j\}_{j\in S}$ as a set of CL's encryption keys, define $\mathcal{R}_{\mathsf{C\text{-}E}} = \{(\{\mathcal{P}_j, \mathcal{C}_j\}_{j\in S}, (\{a_j, a'_j\}_{j\in S}, \rho)) : \forall\ j \in S,\ \mathcal{C}_j = \mathsf{CL.Enc}(ek_j, a_j, \rho), \mathcal{P}_j = a_jG + a'_jH\}$.

For $\mathcal{R}_{\mathsf{C\text{-}E}}$, we can write it as

$$\mathcal{R}_{\mathsf{C\text{-}E}} = \left\{ \begin{array}{c} (\{\mathcal{C}_j = (\mathfrak{R}, \mathfrak{B}_j), \mathcal{P}_j\}_j, \\ (\{a_j, a'_j\}_j, \rho \in \mathcal{D}_q)) \end{array} \middle| \begin{array}{c} \mathfrak{R} = \mathfrak{g}_q^\rho, \mathfrak{B}_j = ek_j^\rho \mathfrak{f}^{a_j} \\ \mathcal{P}_j = a_jG + a'_jH,\ \forall j \in S \end{array} \right\}.$$

By generating $\{e_j\}_{j\in S} \leftarrow \mathsf{H_{fs}}(\{\mathcal{C}_j, \mathcal{P}_j\}_{j\in S})$ as challenges, the prover and verifier can aggregate them into $ek = \prod_{j\in S} ek_j^{e_j}$, $\mathcal{B} = \prod_{j\in S} \mathcal{B}_j^{e_j}$ and $\mathcal{P} = \sum_{j\in S} e_j\mathcal{P}_j$. With $a = \sum_{j\in S} e_ja_j \mod q, a' = \sum_{j\in S} e_ja'_j \mod q$, the above relation is compressed into

$$\mathcal{R}^* = \left\{ \begin{array}{c} ((\mathfrak{R}, \mathfrak{B}), \\ (a, a' \in \mathbb{Z}_q, \rho \in \mathcal{D}_q)) \end{array} \middle| \begin{array}{c} \mathfrak{R} = \mathfrak{g}_q^\rho, \mathfrak{B} = ek^\rho \mathfrak{f}^a \\ \mathcal{P} = aG + a'H \end{array} \right\}.$$

For the relation $\mathcal{R}^*$, we refer the readers to [59, Figure 5] for the concrete construction of its proof system.

Appendix C. Realizing Used Functionalities

C.1. Realizing $\mathcal{F}_{\mathsf{wPDRG}}$ and $\mathcal{F}_{\mathsf{DKG}}$

For a weak version of the PDRG protocol, the parties will abort when malicious behavior is found. We present an implementation where Pedersen commitments [53] are used to detect the presence of incorrect shares. Let the polynomial degree be $d = t+\ell-1$, with the threshold t and the packing parameter ℓ. The two-round Π_{wPDRG} protocol is as follows.

- Round 1: Each party P_i does:
  1. pick two random d-degree polynomials over $\mathbb{Z}_q$:
  $$f_i(X) = a_0 + a_1X + a_2X^2 + \cdots + a_dX^d$$
  $$f'_i(X) = a'_0 + a'_1X + a'_2X^2 + \cdots + a'_dX^d$$
  2. generate shares $k_{i,j} = f_i(j), k'_{i,j} = f'_i(j) \in \mathbb{Z}_q^2$ for each $j \in [n]$
  3. generate the polynomial commitment: $\mathcal{P}_{i,\omega} = a_{i,\omega}G + a'_{i,\omega}H \in \mathbb{G}$ for each $\omega \in [0, d]$
  4. send $(\mathsf{prove}, \{\mathcal{P}_{i,\omega}\}_{\omega\in[0,d]}, \{a_{i,\omega}, a'_{i,\omega}\}_{\omega\in[0,d]})$ to $\mathcal{F}_{\mathsf{zk_{ped}}}$
  5. send $(k_{i,j}, k'_{i,j})$ to P_j for $j \in [n] \setminus \{i\}$.
- Round 2: After receiving $k_{j,i}, k'_{j,i}$ from each P_j and $(\mathsf{proved}, \{\mathcal{P}_{j,\omega}\}_{\omega\in[0,d]}, j)$ from $\mathcal{F}_{\mathsf{zk_{ped}}}$ for $j \in [n]$, P_i does:
  1. for each $j \in [n] \setminus \{i\}$, check $k_{j,i}G + k'_{j,i}H \stackrel{?}{=} \sum_{\omega=0}^{d} i^\omega \mathcal{P}_{j,\omega}$, if it fails, abort the protocol
  2. compute $k_i = \sum_{j\in[n]} k_{j,i} \mod q$, $R_i = k_iG$, $k'_i = \sum_{j\in[n]} k'_{j,i} \mod q$, and $R'_i = k'_iH$
  3. generate the DL proof $\tau_i \leftarrow \mathsf{Prove_{DL}}((R'_i, H), k'_i)$.
  4. send (R_i, τ_i) to each $P_j,\ j \neq i$.
- Output: For each $\omega \in [0, d]$, compute $\mathcal{P}_\omega = \sum_{i\in[n]} \mathcal{P}_{i,\omega}$. For each $j \in [n]$, compute $R'_j = \sum_{\omega=0}^{d} j^\omega \mathcal{P}_\omega - R_j$ and verify τ_j. If τ_j is not valid for some $j \in [n]$, abort the protocol. Otherwise, let $\mathbf{R} = [R_i]_{i\in[n]}^\mathsf{T}$ and assemble ℓ EC points: for each $\nu \in [0, \ell-1]$, $R^{(\nu)} = \boldsymbol{\lambda}_{[n]}^{(-\nu)}\mathbf{R}$. Output $((R^{(0)}, \cdots, R^{(\ell-1)}), (R_1, \cdots, R_n))$.

Theorem 3. *The protocol Π_{wPDRG} realizes $\mathcal{F}_{\mathsf{wPDRG}}$ in $\mathcal{F}_{\mathsf{zk_{ped}}}$-hybrid model against a malicious static adversary corrupting $t \leq n - \ell$ parties.*

Due to space constraints, we omit the proof. It closely follows the key generation phase of threshold ECDSAs [38], [51], with the addition of packed secret sharing.

The difference between $\mathcal{F}_{\mathsf{DKG}}$ and $\mathcal{F}_{\mathsf{wPDRG}}$ is that $\mathcal{F}_{\mathsf{DKG}}$ shares the signing key x in the packed form, while $\mathcal{F}_{\mathsf{wPDRG}}$ shares ℓ random values. Here, we only present the differences in their realizations:

1) In Round 1, P_i picks the random polynomials $f_i(X), f'_i(X)$ such that $f_i(0) = f_i(-1) = \cdots = f_i(-\ell+1)$ and $f'_i(0) = f'_i(-1) = \cdots = f'_i(-\ell+1)$
2) In Round 2, after receiving $\{\mathcal{P}_{j,\omega}\}_{\omega\in[0,d]}$ for each $j \in [n]$, P_i checks
$$\mathcal{P}_{j,0} \stackrel{?}{=} \sum_{\omega=0}^{d} (-1)^\omega \mathcal{P}_{j,\omega} \stackrel{?}{=} \cdots \stackrel{?}{=} \sum_{\omega=0}^{d} (-\ell+1)^\omega \mathcal{P}_{j,\omega}.$$
If it fails, abort the protocol.
3) In Output, output $X = \boldsymbol{\lambda}_{[n]}^{(0)}[X_1, X_2, \cdots, X_n]^\mathsf{T}$.

C.2. Realizing $\mathcal{F}_{\mathsf{sPDRG}}$ and $\mathcal{F}_{\mathsf{sDKG}}$

The Π_{wPDRG} construction cannot be publicly verifiable to identify a malicious party. More precisely, only party P_i can check the correctness of the share $k_{j,i}$ received from P_j. To solve it, we hope that each party can verify the correctness of $k_{j,i}$ from the revealed $k_{j,i}$'s ciphertext. Recently, Cascudo and David [16] used CL homomorphic encryption to build a publicly verifiable (packed) secret sharing (PVSS). We borrow it to realize $\mathcal{F}_{\mathsf{sPDRG}}$. Due to the limited space, we refer the readers to [16] for its concrete construction.

Unlike the realization of $\mathcal{F}_{\mathsf{sPDRG}}$, the protocol for $\mathcal{F}_{\mathsf{sDKG}}$ should let each party share a polynomial $f_{\mathbf{x}_i}(\cdot)$ such as

$$f_{\mathbf{x}_i}(0) = f_{\mathbf{x}_i}(-1) = \cdots = f_{\mathbf{x}_i}(-\ell+1). \tag{6}$$

Thus, we need a PVSS, where the verification will consist of two parts:

1. each party's share comes from the same polynomial, whose degree is no larger than $t + \ell - 1$;
2. the polynomial satisfies eq. (6).

The construction of $\mathcal{F}_{\mathsf{sPDRG}}$ only covers condition 1. However, checking condition 2 upon CL ciphertexts requires $O(n^2)$ operations over the class group.

Instead, we let the dealer not only broadcast encrypted shares but also the Pedersen commitments of the shares. Both conditions can be checked on the Pedersen commitments over the EC group. It is clear that operations in EC groups are much more efficient than those in class groups. The dual-code technique of checking condition 1 from [15], translated to the Pedersen commitment, is as follows: if committed sharing of $f_{\mathbf{x}}(\cdot)$ is correct, then for any random $(n-t-\ell-1)$-degree polynomial $p(\cdot)$, we have $\sum_{j\in[n]} p(j)\cdot v_j \cdot \mathsf{Com}(f_{\mathbf{x}}(j)) = \mathsf{Com}(0)$ for v_j as a predefined public factor. Then condition 2 can be checked via the equality $\mathsf{Com}(f_{\mathbf{x}}(0)) = \cdots = \mathsf{Com}(f_{\mathbf{x}}(-\ell+1))$, which can be generated with committed shares using Lagrange interpolation.

Another key part of our approach is proving the consistency between each encrypted share and its corresponding committed share. Fortunately, these proofs can be aggregated into a single constant-size proof. Inspired by [16], [17], [46], we leverage the same randomness for CL ciphertexts in the multi-receiver setting to make aggregation feasible. With the help of a broadcast channel, we give the construction Π_{sDKG}.

- Setup: P_i establishes a PKI setup with:
  1. CL's key pair $sk_i \leftarrow\!\$\ \mathcal{D}_q, pk_i = \mathfrak{g}_q^{sk_i}$
  2. send $(\mathsf{prove}, pk_i, sk_i)$ to $\mathcal{F}_{\mathsf{zk}_{\mathsf{key}}}$
  3. if $\mathcal{F}_{\mathsf{zk}_{\mathsf{key}}}$ broadcasts $(\mathsf{proved}, pk_i, i)$, P_i is accepted to participate in the protocol.
- Round 1: P_i (as a dealer) picks two degree-$(t+\ell-1)$ polynomials $f_{\mathbf{x}_i}(\cdot), f'_{\mathbf{x}_i}(\cdot)$ satisfying eq. (6) and then shares them:
  1. generate shares $m_j = f_{\mathbf{x}_i}(j), m'_j = f'_{\mathbf{x}_i}(j)$ for each $j \in [n]$
  2. $\mathcal{P}_{i,j} = m_j G + m'_j H \in \mathbb{G}$ for each $j \in [n]$
  3. $\mathfrak{R}_i = \mathfrak{g}_q^{\rho}$ with $\rho \leftarrow\!\$\ \mathcal{D}_q$
  4. $\mathcal{C}_{i,j} = (\mathfrak{R}_i, pk_j^{\rho}\mathfrak{f}^{m_j})$ for each $j \in [n]$
  5. the aggregated Commit-Enc consistency proof
  $$\pi_i \leftarrow \mathsf{Prove}_{\mathsf{C\text{-}E}}((\{\mathcal{P}_{i,j}, \mathcal{C}_{i,j}\}_j), (\{m_j, m'_j\}_j, \rho))$$
  6. broadcast $(\{\mathcal{P}_{i,j}, \mathcal{C}_{i,j}\}_j, \pi_i)$.
- Round 2: After receiving $(\{\mathcal{P}_{j,k}, \mathcal{C}_{j,k}\}_k, \pi_j)$ for each $j \neq i$, P_i first verifies π_j, then check the correctness of $\{\mathcal{P}_{j,k}\}_k$:
  1. with $\vec{\mathcal{P}} = [\mathcal{P}_{j,1}, \cdots, \mathcal{P}_{j,n}]^{\mathsf{T}} \in \mathbb{G}^n$,
  $$\boldsymbol{\lambda}_{[n]}^{(0)}\vec{\mathcal{P}} \stackrel{?}{=} \boldsymbol{\lambda}_{[n]}^{(-1)}\vec{\mathcal{P}} \stackrel{?}{=} \cdots \stackrel{?}{=} \boldsymbol{\lambda}_{[n]}^{(-\ell+1)}\vec{\mathcal{P}}$$
  2. with $p(\cdot)$ as a random degree-$(n-t-\ell-1)$ polynomial and $v_j = \prod_{\mu\in[n]\setminus\{j\}}(\mu - j)^{-1} \in \mathbb{Z}_q$,
  $$[p(1)v_1, p(2)v_2, \cdots, p(n)v_n]\cdot\vec{\mathcal{P}} \stackrel{?}{=} O^{12}.$$

  Let Q be the set of j for whom the sharing has passed the verification. For each $i \in Q$, P_i does:
  1. parse $\mathcal{C}_{j,i} = (\mathfrak{R}_j, \mathfrak{B}_{j,i})$ for each $j \in Q$
  2. assemble the ciphertext $\mathfrak{R} = \prod_{j\in Q}\mathfrak{R}_j, \mathfrak{B}_i = \prod_{j\in Q}\mathfrak{B}_{j,i}$ and decrypt it $x_i = \mathsf{Dlog}(\mathfrak{B}_i/\mathfrak{R}^{sk_i})$
  3. generate $X_i = x_i G \in \mathbb{G}$ and the DL-Dec consistency proof $\pi'_i \leftarrow \mathsf{Prove}_{\mathsf{DL\text{-}D}}(((\mathfrak{R}, \mathfrak{B}_i, X_i), (x_i, sk_i)))$
  4. broadcast $(X_i \in \mathbb{G}, \pi'_i)$.
- Output: Output $X = \boldsymbol{\lambda}_Q^{(0)}[X_i]_{i\in Q}^{\mathsf{T}}$ if $|Q| \geq t + \ell$.

Theorem 4. *Under the DDH-† and RO_C assumptions, the protocol Π_{sDKG} realizes $\mathcal{F}_{\mathsf{sDKG}}$ in the $\mathcal{F}_{\mathsf{zk}_{\mathsf{key}}}$-hybrid model against a malicious static adversary corrupting $t \leq \frac{n-\ell}{2}$.*

The proof is similar to the proof of [16, Theorem 8]. Due to space constraints, we omit it.

C.3. Realizing $\mathcal{F}_{\mathsf{MtA}}$ and $\mathcal{F}_{\mathsf{MtAwPC}}$

There have been many constructions of realizing $\mathcal{F}_{\mathsf{MtA}}$. We abstract CL-based MtA from [19].

- Input: P_A takes $a \in \mathbb{Z}_q$ as input; P_B takes $b \in \mathbb{Z}_q$ as input.
- Round 1: P_A encrypts a to generate $\mathcal{C} \leftarrow \mathsf{CL.Enc}(pk_A, a, \rho)$ with $\rho \leftarrow\!\$\ \mathcal{D}_q$ as the randomness, and sends $(\mathsf{prove}, \mathcal{C}, a, P_B)$ to $\mathcal{F}_{\mathsf{zk}_{\mathsf{enc}}}$.
- Round 2: After receiving $(\mathsf{proved}, \mathcal{C}, P_A)$ from $\mathcal{F}_{\mathsf{zk}_{\mathsf{enc}}}$, P_B picks $\beta \leftarrow\!\$\ \mathbb{Z}_q, \rho' \leftarrow\!\$\ \mathcal{D}_q$ and computes $\mathcal{C}' \leftarrow b \odot \mathcal{C} \oplus \mathsf{CL.Enc}(pk_A, \beta, \rho')$. Send $(\mathsf{prove}, (\mathcal{C}, \mathcal{C}'), (b, \beta), P_A)$ to $\mathcal{F}_{\mathsf{zk}_{\mathsf{aff}}}$.
- Output: After receiving $(\mathsf{proved}, (\mathcal{C}, \mathcal{C}'), P_B)$ from $\mathcal{F}_{\mathsf{zk}_{\mathsf{aff}}}$, P_A decrypts $\alpha \leftarrow \mathsf{CL.Dec}(\mathcal{C}', sk_A)$. P_A takes $\alpha \in \mathbb{Z}_q$ as output; P_B takes $-\beta \in \mathbb{Z}_q$ as output.

$\mathcal{F}_{\mathsf{MtA}}$ does not involve checking the correctness of inputs (a or b), but this is often required in threshold ECDSA. More strictly, in constructions aimed at robustness, we need $\mathcal{F}_{\mathsf{MtAwPC}}$, of which the public checking helps participants identify the cheating parties publicly. For threshold ECDSA, we only need to check the correctness of P_B's input, with $B = bG \in \mathbb{G}$ as the public input. With the help of a broadcast channel, the two-round Π_{MtAwPC} is as follows.

- Round 1: Same as the above.
- Round 2: $\mathcal{F}_{\mathsf{zk}_{\mathsf{aff}}}$ is replaced by $\mathcal{F}_{\mathsf{zk}_{\mathsf{aff\text{-}DL}}}$, where proved instructions are broadcast to all parties.
- Public Checking: With common input $B \in \mathbb{G}$, each party P_i (including P_A) would receive $(\mathsf{proved}, (\mathcal{C}, \mathcal{C}', B), P_B)$ from $\mathcal{F}_{\mathsf{zk}_{\mathsf{aff\text{-}DL}}}$. If P_i does not receive it, it records P_B as a malicious party.
- Output: P_A decrypts $\alpha \leftarrow \mathsf{CL.Dec}(\mathcal{C}, sk_A)$. P_A takes $\alpha \in \mathbb{Z}_q$ as output; P_B takes $-\beta \in \mathbb{Z}_q$ as output.

The difference between Π_{MtAwPC} and Π_{MtA} is that Π_{MtAwPC} requires each party to verify the zero-knowledge proofs rather than only P_A or P_B in Π_{MtA}.

Theorem 5. *Under the HSM assumption, the protocol Π_{MtAwPC} realizes $\mathcal{F}_{\mathsf{MtAwPC}}$ in the $(\mathcal{F}_{\mathsf{zk}_{\mathsf{enc}}}, \mathcal{F}_{\mathsf{zk}_{\mathsf{aff\text{-}DL}}})$-hybrid model.*

Proof. The proof proceeds in two cases: adversary $\mathcal{A}$ corrupts P_A, and corrupts P_B.

12. $O \in \mathbb{G}$ is the infinity point with $\mathsf{Dlog}(O) = 0$

Simulating P_B for corrupted P_A. In Round 1, the simulator Sim receives a on behalf of $\mathcal{F}_{\mathsf{zk}_{\mathsf{enc}}}$. Then Sim sends $(\mathsf{input}, sid, P_A||P_B, a)$ to $\mathcal{F}_{\mathsf{MtAwPC}}$ on behalf of P_A. After receiving $(\mathsf{output}, sid, P_A||P_B, \alpha)$ from $\mathcal{F}_{\mathsf{MtAwPC}}$, Sim simulates the ciphertext $\mathcal{C}' \leftarrow \mathsf{CL.Enc}(pk_A, \alpha, \rho')$ with $\rho' \leftarrow\!\!\$\ \mathcal{D}_q$. Send $\mathcal{C}'$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{zk}_{\mathsf{aff\text{-}DL}}}$. In public checking phase, send $(\mathsf{proved}, (\mathcal{C}, \mathcal{C}', B), P_B)$ to all parties (including $\mathcal{A}$) on behalf of $\mathcal{F}_{\mathsf{zk}_{\mathsf{aff\text{-}DL}}}$.

The main difference between the simulation and a real execution is the generation of $\mathcal{C}'$. In the real world, $\mathcal{C}'$ is the ciphertext of $ab+\beta$ with $\beta \leftarrow\!\!\$\ \mathbb{Z}_q$. This is exactly the same distribution $\mathcal{F}_{\mathsf{MtAwPC}}$ uses to choose α. Thus, the ciphertext $\mathcal{C}'$ seen by the adversary in the simulation is as distributed as in the protocol.

Simulating P_A for corrupted P_B. In Round 1, Sim picks $r \leftarrow\!\!\$\ \mathbb{Z}_q$, generates r's ciphertext $\mathcal{C}$, and sends $\mathcal{C}$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{zk}_{\mathsf{enc}}}$. After receiving (b, β) on behalf of $\mathcal{F}_{\mathsf{zk}_{\mathsf{aff\text{-}DL}}}$ in Round 2, Sim sends $(\mathsf{multiply}, sid, P_A||P_B, b)$ to $\mathcal{F}_{\mathsf{MtAwPC}}$ on behalf of P_B. If $B = bG$, send $(\mathsf{proved}, (\mathcal{C}, \mathcal{C}', B), P_B)$ to all parties on behalf of $\mathcal{F}_{\mathsf{zk}_{\mathsf{aff\text{-}DL}}}$ in public checking phase.

The main difference between the simulation and a real execution is the generation of $\mathcal{C}$. In the real world, $\mathcal{C}$ is the ciphertext of the input value $a \in \mathbb{Z}_q$, while the simulated $\mathcal{C}$ is the ciphertext of a uniformly random $r \leftarrow\!\!\$\ \mathbb{Z}_q$. Under the HSM assumption, the CL encryption has semantic security [21]. Thus, r's ciphertext is indistinguishable from a's ciphertext for the adversary $\mathcal{A}$. □