ARES: Online-Friendly Robust Threshold ECDSA with Amortized Costs

Guofeng Tang[1], Tian Qiu[2], Bowen Jiang[1], Haiyang Xue[1], Meng Hao[1], Guomin Yang[1], and Robert H. Deng[1]

[1] Singapore Management University
tang.guofeng789@gmail.com, bowen.jiang.2024@phdcs.smu.edu.sg, menghao303@gmail.com
{haiyangxue,gmyang,robertdeng}@smu.edu.sg
[2] Digital Trust Centre, Nanyang Technological University qtautumn6@gmail.com

Abstract. Threshold ECDSA has been an active research topic in recent years, driven by its wide-ranging applications, particularly in blockchain domains. In these real-world applications, robustness is a critical requirement. It ensures that a signature is successfully generated as long as $t+1$ honest parties are present, regardless of malicious behavior from others. Existing robust constructions 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 sending communication per party but incurs an expensive online phase. In contrast, the MtA-based approach (e.g., TX25 in S&P'25) is online-friendly, requiring only elliptic-curve group operations during the online phase. However, it has the drawback of requiring $O(n)$ sending communication per party when n parties are involved.
In this work, we propose ARES, a robust threshold ECDSA scheme designed to reduce both communication and computational overhead within the online-friendly MtA framework. To improve the communication efficiency of TX25, we propose *verifiable non-interactive multiplication* (VNIM), a new primitive which endows standard non-interactive multiplication with public verifiability. Simultaneously, we leverage super-invertible matrices and packed secret sharing to amortize the overall costs. Specifically, when setting the packing parameter of packed secret sharing to $\ell = 1$ (i.e., without secret packing), ARES exhibits linear communication complexity while already achieving an approximately 50% improvement over TX25. When configuring $\ell = t/3$ with $t \geq 12$, ARES achieves lower communication overhead than WMC24, requiring a constant ≈ 4.5 KB per party. Furthermore, setting $\ell = t/2$ reduces the amortized communication cost to roughly 3 KB per party. On the other hand, amortizing across ℓ signatures incurs a trade-off by increasing the required party size by ℓ.

Keywords: Threshold ECDSA, Robustness, Verifiable NIM, Packed secret sharing.

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 fewer) parties cannot generate a signature [45]. 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 signatures (and multi-party threshold cryptography in general) is acknowledged by NIST in its ongoing standardization process [10].

Since ECDSA is currently the most widely used signature scheme in cryptocurrencies, many academic studies have focused on the efficient design of threshold ECDSA [37,39,30,23,46]. 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 users.

In recent years, the robustness (also known as guaranteed output delivery in generic MPC [3]) of threshold ECDSA has received increasing attention, as it is a critical requirement for real-world applications [50,49,48,33,41,47]. A robust signing protocol always produces a valid signature even in the presence of malicious behavior, as long as a sufficient number of parties remain semi-honest.

This could prevent Denial-of-Service (DoS) attacks. In contrast, non-robust signing protocols invariably abort in the presence of malicious behavior. While some schemes [12,18] support identifiable abort, enabling detection of misbehaving parties, the typical mitigation is to exclude the cheaters and restart the protocol until it succeeds. However, the total round complexity scales with the number of misbehaving parties, which is unacceptable for time-sensitive applications.

In general, achieving robustness requires a *broadcast* channel, enabling honest parties to consistently identify and exclude malicious parties [19]. Moreover, a robust protocol requires an honest-majority setting ($n \geq 2t+1$ with t as the maximum number of corrupted parties), since $t+1$ honest parties are required to contribute correct shares [17,38], yet it still achieves security with abort under a dishonest majority setting.

Robust Threshold ECDSAs. Threshold ECDSAs with/without robustness can be classified into two categories based on different technical tradeoffs: those based on threshold homomorphic encryption [31,49,33] and those based on multiplicative-to-additive conversion [12,52,25,46]. For further discussion on related work, please refer to Section 1.3.

Threshold Homomorphic Encryption. Linearly Homomorphic Encryption (LHE) is an encryption scheme where, under a public key pk, given the encryptions of a and 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 decryption key is distributed among multiple parties. A secure distributed decryption protocol ensures that the plaintext can be recovered without revealing any additional information.

Using TLHE as a tool, Wong et al. [49] (denoted by WMC24) constructed a robust threshold ECDSA scheme, establishing a framework followed by subsequent works such as [33] and [41]. Since this construction follows the "You-Only-Speak-Once" (YOSO) paradigm of general multi-party computation [11], it benefits from $O(1)$ broadcast communication per party. However, its online phase[3] is relatively costly due to LHE decryption, which is much more expensive than ECDSA operations.

Multiplicative-to-Additive (MtA) Conversion. MtA is a two-party protocol that, given inputs a and b from each party, outputs additive shares α and β, such that $\alpha + \beta = a \cdot b$. It can be efficiently constructed from LHEs [30,15,51] and oblivious transfer (OT) [23] with two rounds of communication. Recently, a single-round MtA has also been developed [9], referred to as Non-Interactive Multiplication (NIM).

Using MtA as a building block, one can construct robust threshold ECDSA schemes, as shown in WMYC23 [50] and TX25 [48]. These schemes offer an efficient online phase, a property referred to as being "online-friendly" [48], which means online computation only involves field operations in $\mathbb{Z}_q$ and group operations in $\mathbb{G}$. However, this efficiency comes at a cost: they require $O(n)$ broadcast communication per party. It is worth noting that Lyu et al. [40] utilized NIM to construct a two-round threshold ECDSA scheme with constant per-party communication cost. Nevertheless, their scheme lacks robustness, which is the focus of this work.

As an exception, Trout++ [42] applies the ROAST transformation [43] to the two-round threshold ECDSA scheme Trout [20], achieving a robust protocol over asynchronous networks. However, it relies on an external robustness wrapper to iteratively filter and collect correct protocol messages until $t+1$ honest shares are gathered, enabling aggregation of a valid signature. This incurs a worst-case round complexity of $O(n)$. In contrast, this paper focuses on threshold signing protocols that achieve intrinsic robustness rather than relying on an external wrapper. Consequently, we omit further discussion of Trout++.

Motivation. As analyzed above and summarized in Table 1, MtA-based robust schemes require linear communication and quadratic computational complexity per party. Despite its asymptotic disadvantages, the MtA method enables online-friendly design [48], making 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 and computational complexity of MtA-based robust schemes, thereby making them competitive with TLHE-based schemes in terms of asymptotic performance?*

[3] Threshold signing protocols can be divided into offline and online phases depending on whether the signing message is required or not.

Table 1: Asymptotic performance (sending communication and computation per party) of **robust** threshold ECDSAs

Schemes	Tools	Off. Comm.	On. Comm.	Off. Comp.	On. Comp.	Rounds	OF[◇]	Party Count
WMYC23 [50]	MtA	$O(n)$	$O(n)$	$O(n^2)$	-	5	✓	$n \geq 2t+1$
WMC24 [49]	TLHE	$O(1)$	$O(1)$	$O(n)$	$O(n)$	4	✗	$n \geq 2t+1$
TX25 [48]	MtA	$O(n)$	$O(n)$	$O(n^2)$	-	3	✓	$n \geq 2t+1$
ARES	VNIM, SIM, PSS	$\frac{O(n)}{\epsilon t}$	$\frac{O(n)}{\epsilon t}$	$\frac{O(n^2)}{\epsilon t}$	-	3	✓	$n \geq 2t+\epsilon t^{\star}$

[◇] The scheme is online-friendly (OF) if the online computation is restricted to field operations in $\mathbb{Z}_q$ and group operations in $\mathbb{G}$.

[⋆] $1/t \leq \epsilon < 1$. Our asymptotic performance is amortized over $\ell = \epsilon t$ signatures.

1.1 Our Contributions

As evidenced by Table 1, we provide an affirmative answer to the above question in this paper by presenting ARES.

Technically, we achieve this by introducing verifiable non-interactive multiplication (VNIM), an enhanced variant of NIM equipped with verifiability. Concretely, whereas the standard NIM splits a product $x_A \cdot x_B$ into additive shares s_A and s_B between two parties, VNIM further enforces that the two parties use their authentic inputs x_A (resp. x_B) and produce verifiable public commitments to the resulting shares, i.e., $S_A = s_A G$ (resp. $S_B = s_B G$). This verifiability underpins the robustness of our scheme: the public points S_A and S_B allow parties to validate partial signatures and consistently identify malicious parties. We believe VNIM may be of independent interest for other MPC applications.

Although employing VNIM has improved bandwidth efficiency compared to MtA with public checking, as used in TX25 [48], the per-party communication complexity for sending remains linear. To overcome this limitation, we leverage super-invertible matrices (SIM) and packed secret sharing (PSS) to amortize the communication cost to $O(1)$ per signature. As a trade-off, utilizing packed secret sharing with a packing parameter ℓ requires an additional ℓ parties. As another technical contribution, to render the DKG phase (which shares the signing key x) compatible with packed secret sharing while preserving public verifiability, we propose a specialized Publicly Verifiable Packed Secret Sharing scheme, termed PVSSeq. In this scheme, all packed secrets are identical—specifically, they represent the same signing key x. We believe this PVSSeq construction may also be of independent interest.

Table 2 illustrates the concrete communication performance of ARES. At $\ell = 1$ (no secret packing), ARES scales linearly but still cuts the communication overhead by $\approx 50\%$ compared to TX25 [48]. For $\ell = t/3$ ($t \geq 12$), ARES outperforms WMC24 [49] by attaining a constant overhead of ≈ 4.5 KB per party, which further decreases to ≈ 3 KB when $\ell = t/2$. Table 3 demonstrates that ARES, like TX25, is online-friendly, achieving an online computational efficiency that significantly outperforms WMC24.

1.2 Technical Overviews

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 $\mathbb{G}$ with order q, x is the secret key, k is the secret nonce, r is the x-coordinate of kG, and $m = \mathsf{H}(\mathsf{msg})$ with a hash function $\mathsf{H}(\cdot) : \{0,1\}^* \to \mathbb{Z}_q$. The primary challenge in threshold ECDSA is computing k^{-1} and $k^{-1}x$ when parties hold only additive shares (x_i, k_i). To compute k^{-1}, protocols introduce a second secret nonce γ and compute the signature component s via $s = \frac{\gamma(m + r \cdot x)}{\gamma k} \bmod q$.

Assume each party P_i holds a t-out-of-n share x_i of the secret key x. Let S be the set of active signers; we require $|S| \geq 2t+1$ for robustness (this is a necessary requirement according to [17,38]). We start by reviewing the basic structure of MtA-based robust threshold ECDSA in TX25 [48], which is based on CL homomorphic encryption from the class group [16].

- Offline Phase (Rounds 1 & 2): The parties execute a Distributed Randomness Generation (DRG) protocol to jointly generate $R = kG$, such that each party obtains a t-out-of-n share k_i of k. Then, each pair of parties execute MtA calls on $\gamma_i k_j$ and $\gamma_i x_j$ where γ_i is picked by each party, for each $i, j \in S$. Note that MtA with public checking (i.e., the additive share of each $\gamma_i k_j$ or $\gamma_i x_j$ needs to be checked) is required for robustness.
- Online Phase (Round 3): For a hashed message m, each party broadcasts the shares generated via MtA protocols. These include the shares of γk and the shares of $\gamma(m+rx)$ (linearly computed from the shares of $\gamma x, \gamma$, the hashed message m, and the nonce r), as well as the corresponding zero-knowledge proof. The final signature is then assembled from $t+1$ honest shares.

We divide this protocol into three components and provide the complexity for each component as follows:

1. DRG component requires $O(n)$ sending communication and $O(n^2)$ computation per party.
2. The process of MtA with public checking has the same complexity as the DRG.
3. The online phase also requires $O(n)$ communication per party. Note that TX25 is online-friendly, with online operations that are standard elliptic-curve group operations, which are substantially more efficient than the class-group operations required in the offline phase. Thus, we ignore its online computation.

Clearly, we need to reduce the complexity of each component.

Improve DRG using Super-Invertible Matrix (SIM). Our intuition is to leverage the concept of amortization. Specifically, for the DRG component required for robustness, complexity could be reduced if a single execution could batch-generate multiple values of $R = kG$. In the baseline construction [48], each party picks a random s_i and shares it via publicly verifiable secret sharing (PVSS). The final joint randomness k is defined as $\sum_i s_i$. Here, k is random if only a single s_i is random, so a single honest party is enough.

Intuitively, if we have many honest parties, we can obtain many random nonces k. Fortunately, there are at least $t+1$ honest parties for robustness. Let $\hat{\ell} = n - t$ and $\boldsymbol{\Psi} = [\psi_i^{(\nu)}]$ be a $\hat{\ell} \times n$ Super-Invertible Matrix (SIM), i.e., each $\hat{\ell} \times \hat{\ell}$ sub-matrix is invertible (e.g., a Vandermonde matrix) [32,4]. Then, we can assemble $\hat{\ell}$ random and independent nonces $k^{(\nu)} = \sum_i \psi_i^{(\nu)} s_i, \forall \nu \in [\hat{\ell}]$, since $\boldsymbol{\Psi}$'s $\hat{\ell} \times \hat{\ell}$ sub-matrix corresponding to the rows of these $\hat{\ell}$ honest parties is invertible. In this manner, the $O(n)$ communication overhead incurred by each party's PVSS invocation can be amortized across $\hat{\ell}$ independent instances of R.

Improve MtA using Verifiable NIM and Packed Secret Sharing. For the multiplicative-to-additive (MtA) conversion (i.e., converting each product $\gamma_i k_j, \gamma_i x_j$ into additive shares), inspired by [40], one could employ Non-Interactive Multiplication (NIM) [9] to improve communication and round efficiency. However, the construction in [40] is not robust.

The core challenge lies in achieving public verifiability with NIM. When parties perform further computations on their local additive shares (such as computing and publishing a signature share in the online phase derived from an NIM share), other participants cannot consistently verify its correctness. Without public verifiability, participants are forced to resort to an inefficient trial-and-error approach: guessing at a subset of honest parties, reconstructing the signature, checking its validity, and repeating the process if the check fails. In the worst case, honest parties may require exponential time (in the number of parties) to produce an output, rendering the approach impractical.

To address this issue, we propose a new primitive termed Verifiable Non-Interactive Multiplication (VNIM). Taking two parties P_i with input γ_i and P_j with input k_j as an example, we first recall the standard NIM protocol: P_i publishes the encoding of its input, ec_{γ_i}, generated via the Encode algorithm. Using P_j's encoding ec_{k_j} alongside its own input γ_i, P_i can locally compute its additive share s_A via the Decode algorithm. Symmetrically, P_j obtains s_B, such that $s_A + s_B = \gamma_i k_j$.

In the syntax of VNIM, the Encode algorithm is replaced by a verifiable counterpart, $\mathsf{VEncode}$, where ec_{γ_i} is accompanied by a zero-knowledge proof (ZKP) certifying the validity of γ_i. Crucially, and representing our core technical contribution, Decode is likewise upgraded to a verifiable decoding algorithm, $\mathsf{VDecode}$. Here, the public counterpart of s_A (e.g., $S_A = s_A G$) is revealed along with a ZKP proving its correctness. The central hurdle lies in designing this ZKP. The standard decoding procedure relies on a coset-labeling function $\phi(\cdot)$ over the underlying group structure (e.g., class groups or the Paillier group $\mathbb{Z}_{N^2}^*$), which maps every element in a coset to a designated canonical representative $\mathfrak{h}^*$; the additive share s_A is then derived from this representative and the private input. A naive approach to proving the correctness of S_A would be to prove that its discrete logarithm equals $\mathsf{Decode}(\gamma_i, \mathsf{ec}_{k_j})$. However, Decode inherently involves non-linear operations within $\phi(\cdot)$, such as projecting a group element onto a subgroup and lifting the resulting element back to the ambient group, rendering a direct ZKP hopelessly impractical.

We construct an efficient VNIM instantiation based on class groups (which naturally extends to the Paillier group $\mathbb{Z}_{N^2}^*$ and the RSA group $\mathbb{Z}_N^*$), motivated by the fact that the plaintext space in class groups naturally aligns with the curve order of ECDSA. Our starting observation is that the canonical representative $\mathfrak{h}^*$ can actually be made public, as it is a common intermediate value computed by both P_i and P_j. We then decouple the proof into two parts.

1. Proving the correctness of the shares: Proving that the discrete logarithm of S_A (resp. S_B) is correctly derived from $\mathfrak{h}^*$ and P_i's (resp. P_j's) private input. Crucially, this step constitutes a strictly linear relation (formally defined as $\mathcal{R}_{\mathsf{outA}}$ and $\mathcal{R}_{\mathsf{outB}}$ in Section 3.1).
2. Proving the validity of $\mathfrak{h}^*$: Showing that the published $\mathfrak{h}^*$ is well-formed.

The second part is rendered straightforward by another key insight: first, the canonical property of $\mathfrak{h}^*$ is publicly verifiable simply by checking that $\phi(\mathfrak{h}^*) = \mathfrak{h}^*$; second, the fact that $\mathfrak{h}^*$ originates from the correct coset is already subsumed by the proof in 1. Furthermore, we propose a multi-party optimization for VNIM that enables aggregating multiple proofs into a single, compact proof via the $\mathsf{VDecode}_B^m$ interface. This optimization directly benefits our threshold ECDSA scheme, which inherently deploys VNIM across a multi-party setting.

Although combining VNIM with super-invertible matrices (SIM) significantly improves communication efficiency over TX25, the per-party communication complexity remains linear due to broadcasting $\mathfrak{h}^*$ in each MtA execution. To further enhance amortized efficiency, we incorporate Packed Secret Sharing (PSS), which enables packing multiple secrets into a single shared polynomial. More concretely, rather than embedding only a single secret at $f(0)$, additional secrets are encoded at evaluation points $f(-1), f(-2), \ldots, f(-(\ell-1))$[4], while each party's share remains $f(i)$. To protect these ℓ packed secrets against up to t corrupted parties, the polynomial $f(\cdot)$ is chosen to have degree $t + \ell - 1$. Standard Shamir's secret sharing corresponds to the special case where $\ell = 1$.

However, we require a special packed secret sharing scheme for the signing key x during the Distributed Key Generation (DKG) phase. Note that we aim to generate signatures for a fixed public key $X = xG$. To ensure compatibility with multiple k's packed into a single PSS polynomial, the secret key x must be shared via a polynomial $f_x(\cdot)$ such that:

$$f_x(0) = f_x(-1) = \cdots = f_x(-\ell + 1) = x. \tag{1}$$

At the same time, for robustness, such shares are generated using PVSS [50,48], where the public verifiability ensures that malicious shares can be consistently filtered out without additional interaction. That is, our construction requires a PVSS that can specifically verify the equality in eq. (1), a feature missing from existing PVSS constructions [14,34]. Crucially, we require this verification to be achieved without extra rounds or substantial computational costs.

To solve this, we introduce PVSSeq, a PVSS scheme tailored to satisfy the verification requirements of eq. (1), thereby enabling distributed key generation for our robust scheme. Our approach extends the PVSS construction in [14] by adding validation of eq. (1). A detailed description is provided in Section 3.2.

[4] This requires that the evaluation domain $\{0, -1, \ldots, -(\ell-1)\}$ and the parties' indices $\{1, 2, \ldots, n\}$ be disjoint.

Improving online efficiency by rewriting the signing equation. In the online phase of robust threshold ECDSA schemes, signature shares typically incur a communication complexity of $O(n)$. To reduce this overhead, we aim to make the broadcasted signature shares reusable across ℓ distinct signature instances, thereby amortizing the per-instance online communication down to a constant. To this end, our online protocol requires a packed secret sharing for the message-dependent term $m + rx$, packing ℓ messages $\{m^{(\nu)}\}_{\nu\in[\ell]}$ and ℓ nonces $\{r^{(\nu)}\}_{\nu\in[\ell]}$, where each $r^{(\nu)}$ is derived from the public group element $k^{(\nu)}G$ generated in the offline phase. Since both $\{m^{(\nu)}\}$ and $\{r^{(\nu)}\}$ are public, one can easily interpolate them into $(\ell-1)$-degree polynomials $f_m(\cdot)$ and $f_r(\cdot)$ in packed form. However, if we were to directly compute the standard sharing polynomial $f_m + f_r \cdot f_x$, multiplying f_r (of degree $\ell-1$) by f_x (of degree $t+\ell-1$) would yield a polynomial of degree $t + 2\ell - 2$. This degree inflation is undesirable, as it would require at least $t + 2\ell - 1$ honest signers to successfully reconstruct the signature. This limitation has already been observed in recent work [27], which used packed secret sharing to improve the non-robust threshold ECDSA [25].

To circumvent this limitation, we rewrite the s-component algebraically: $s = \frac{\gamma(m+r\cdot x)}{\gamma k} = r \cdot \frac{\gamma(mr^{-1}+x)}{\gamma k}$. By defining $\hat{r} = mr^{-1}$, this transformation converts the multiplicative interaction between r and x into an additive relation $\hat{r} + x$, enabling us to replace polynomial multiplication with polynomial addition. Specifically, the ℓ public values of $\hat{r}$ can be interpolated into a polynomial $p(\cdot)$ of degree $\ell-1$. Consequently, the degree of the resulting sharing polynomial $p + f_x$ remains $t + \ell - 1$. In this manner, we eliminate the need for extra signers while maintaining an optimal reconstruction threshold. Please refer to Figure 3 for the concrete workflow of our online phase.

Put it all together. Ultimately, we construct ARES, which successfully amortizes the offline communication and computational overhead down to $O(n/\epsilon t)$ and $O(n^2/\epsilon t)$ per signature, respectively, where $\ell = \epsilon t$ $(0 < \epsilon < 1)$ denotes the packing parameter of the PSS technique. Furthermore, we reduce the amortized online communication cost to $O(n/\epsilon t)$ per party. While incorporating PSS necessitates increasing the total party count from $n = 2t+1$ to $n = 2t+\ell$, this trade-off makes the scheme particularly advantageous in large-scale settings where n is large. Notably, NIST's call for multi-party threshold schemes emphasizes interest in such large-scale deployments [10, Table 16].

Discussions about the limitations. On the other hand, the online amortizing requires the online phase to be executed in batches, necessitating that ℓ messages be signed simultaneously. 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 [7]. When receiving multiple withdrawal requests within a short period, the exchange processes them in batches.

In application scenarios where signing is restricted to one-by-one, ARES can transition to a specialized mode that preserves offline optimizations while maintaining a one-by-one signing approach during the online phase. Refer to Remark 3 for details. Although this mode sustains linear complexity in online communication, the actual payload remains remarkably compact, requiring only $\approx 0.065n$ KB for 128-bit security.

1.3 Related Works

In the honest-majority setting, the multiplications required in threshold ECDSA can be handled more efficiently, albeit without robustness. Roughly speaking, given secrets a and b, let $[a]_t$, $[b]_t$ denote degree t Shamir shares of a, b respectively. Shamir secret sharing supports a multiplicative operation that yields a degree-$2t$ sharing of the product $[ab]_{2t} = [a]_t \cdot [b]_t$ when $t < n/2$, where each party locally computes the product of its shares [22]. Katz and Urban [36] present state-of-the-art work in this line, achieving high performance without relying on any additional public-key cryptographic primitives. On the other hand, such protocols inherently require an honest-majority assumption even to achieve security with abort, whereas our scheme guarantees security with abort in the dishonest-majority setting.

In comparison with OT-based constructions [24,26], our CL-based scheme presents a trade-off: while the computational execution time is approximately two orders of magnitude slower, the communication overhead is reduced by a corresponding two orders of magnitude. For OT-based constructions, achieving robustness is inherently difficult, as public verifiability in OT-based protocols is challenging to implement efficiently.

Recently, two threshold ECDSA schemes with two rounds have been proposed, [20] and [40], without considering robustness. Both of them utilize CL encryption and achieve constant per-party sending costs. Concretely, the scheme of [40] leverages NIM to handle distributed multiplication and achieves very high communication efficiency; however, its signing protocol produces a biased nonce k, which limits its security analysis to a game-based model. In comparison, by introducing an additional round, our scheme ensures that the generated nonce k is unbiased and distributed across parties under a true threshold structure—a fundamental prerequisite for both attaining robustness and establishing simulation-based security. Simultaneously, we augment NIM with public verifiability, thereby enabling native robustness. Trout [20] utilizes exponent VRF (eVRF) [8] and scaled decryption to achieve a two-round construction. However, conferring intrinsic robustness onto Trout without resorting to an external wrapper is non-trivial. In particular, a natively robust variant would require a key-homomorphic eVRF, whose efficient construction remains an open problem.

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.

Definition 1 (Super-invertible matrix, SIM [32,4]). *An a-by-b ($b > a$) matrix M is super-invertible if for any index sets $S \subseteq [b]$ with $|S| = a$, the sub-matrix $\hat{M}$ is invertible, where $\hat{M}$ denotes the matrix consisting of the columns $j \in S$ of M.*

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 [21] 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 $R' = s^{-1}(\mathsf{H}(\mathsf{msg})G + rX)$.

It is well-known that for every valid signature (r, s), the pair $(r, -s)$ is also valid. To ensure signature uniqueness, it is standard to mandate the smaller s-value.

2.2 Packed Secret Sharing

Let $\{P_i\}_{i\in[n]}$ be all parties and $\mathbf{x} = [x^{(1)}, \cdots, x^{(\ell)}] \in \mathbb{Z}_q^\ell$ be all secrets that are packed in one sharing.

- $\mathsf{PackSS}(\mathbf{x}, t) \to [\mathbf{x}]_{t+\ell-1}$: A dealer picks a random degree-$(t + \ell - 1)$ polynomial $f(\cdot) \in \mathbb{Z}_q[X]$ such that $f(-\nu + 1) = x^{(\nu)}, \nu \in [\ell]$, 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 $\hat{n} = |S| \geq t+\ell$ and $S = \{i_1, \cdots, i_{\hat{n}}\} \subseteq [n]$, denote the Lagrange coefficient corresponding to the set S and point $-\nu+1$ as $\lambda_i^{(\nu)} = \prod_{k\in S, k\neq i} \frac{-\nu+1-k}{i-k} \bmod q$ and return

$$\begin{pmatrix} x^{(1)} \\ \vdots \\ x^{(\ell)} \end{pmatrix} = \underbrace{\begin{pmatrix} \lambda_{i_1}^{(1)} & \lambda_{i_2}^{(1)} & \cdots & \lambda_{i_{\hat{n}}}^{(1)} \\ \vdots & \vdots & & \vdots \\ \lambda_{i_1}^{(\ell)} & \lambda_{i_2}^{(\ell)} & \cdots & \lambda_{i_{\hat{n}}}^{(\ell)} \end{pmatrix}}_{\boldsymbol{\Lambda}_S = [\lambda_i^{(\nu)}]_{i\in S}^{\nu\in[\ell]} \in \mathbb{Z}_q^{\ell\times\hat{n}}} \cdot \begin{pmatrix} x_{i_1} \\ x_{i_2} \\ \vdots \\ x_{i_{\hat{n}}} \end{pmatrix}$$

  Otherwise return $\perp$.

The correctness can be derived from the interpolation equation $x^{(\nu)} = f(-\nu+1) = \sum_{i\in S} \lambda_i^{(\nu)} x_i \in \mathbb{Z}_q$ if $|S| \geq t+\ell$. The t-privacy relies on the fact that any t shares from a random degree-$(t+\ell-1)$ polynomial are independent of the ℓ secrets $\mathbf{x}$.

Lemma 1. *The matrix $\boldsymbol{\Lambda}_S = [\lambda_i^{(\nu)}]_{i\in S}^{\nu\in[\ell]} \in \mathbb{Z}_q^{\ell\times\hat{n}}$ is a super-invertible matrix. (The proof is deferred to Appendix B.1.)*

2.3 Publicly Verifiable Secret Sharing

Following [14], we present the syntax of publicly verifiable secret sharing (PVSS) as follows. Owing to space limitations, we defer its security definitions and the construction from [14] to Appendix C.

Definition 2 (PVSS). *A general PVSS should also include the* Reconstruct *algorithm. Since our construction does not require it, we omit it here. It supports packed secret sharing with ℓ as the packed parameter.*

- $\mathsf{PVSS.Setup}(1^\lambda, \mathsf{info}) \to \mathsf{pp}$*: It outputs public parameters* pp*, with the input of the initial information* info *containing information about the number of parties, the reconstruction threshold, and spaces of secrets and shares.*
- $\mathsf{PVSS.KeyGen}(\mathsf{pp}, i) \to (dk_i, ek_i, \pi_{\mathsf{key}_i})$*: Executed by party P_i, the process outputs a decryption and encryption key pair $(dk_i, ek_i) \in \mathcal{R}_{\mathrm{key}}$, along with a proof π_{key_i} which demonstrates that ek_i is a valid public encryption key.*
- $\mathsf{PVSS.Share}(\mathsf{pp}, \{ek_i\}_{i\in[n]}, \mathbf{s}) \to (\{\mathcal{C}_i\}_{i\in[n]}, \pi_{\mathsf{Sh}})$*: On input the secret vector $\mathbf{s} \in \mathbb{Z}_q^\ell$, it outputs "encrypted shares" $\{\mathcal{C}_i\}_{i\in[n]}$ and a proof π_{Sh} asserting their correctness.*
- $\mathsf{PVSS.VerifySharing}(\mathsf{pp}, \{ek_i, \mathcal{C}_i\}_{i\in[n]}, \pi_{\mathsf{Sh}}) \to 0/1$*: It outputs a bit indicating whether the sharing is valid.*

2.4 Castagnos-Laguillaumie Encryption

Class Group. The class group framework [16] is defined by the public parameter $pp_{\mathsf{cl}} = (\tilde{s}, q, \mathfrak{f}, \mathfrak{g}_q, \mathcal{F}, \mathcal{G}^q, \hat{\mathcal{G}})$ where $\hat{\mathcal{G}}$ is an unknown-order class group with order $q\hat{s}$, only the upper bound $\tilde{s}$ of $\hat{s}$ is given. It utilizes $\hat{\mathcal{G}}$'s three subgroups:

- $\mathcal{F} := \langle \mathfrak{f} \rangle$, a unique subgroup of order q with a *deterministic polynomial-time* algorithm $\mathsf{Dlog}(\cdot)$ that returns x on input $\mathfrak{f}^x \in F$;
- $\mathcal{G}^q := \langle \mathfrak{g}_q \rangle$, a subgroup containing all q-th powers in $\mathcal{G}$ with an unknown order s s.t. $s|\hat{s}$;
- $\mathcal{G} := \mathcal{F} \times \mathcal{G}^q = \langle \mathfrak{f}\mathfrak{g}_q \rangle$, with order qs.

The Castagnos-Laguillaumie (CL) encryption works over the class group. 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 $\mathcal{G}^q$, where λ_s is the statistical parameter. The CL scheme and linear homomorphic operations are described below.

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

2.5 Zero-Knowledge Proof

Let $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ be a PPT decidable binary relation. We call w a witness for an instance x if $(x,w) \in \mathcal{R}$, and define the language $\mathcal{L} = \{x | \exists\ w, \text{ s.t. } (x,w) \in \mathcal{R}\}$ as the set of instances x that have a witness w in the relation $\mathcal{R}$. In this work, we use a non-interactive zero-knowledge (NIZK) proof, which is always generated via the Fiat-Shamir [28] (which requires rewinding) or Fischlin transform [29] 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 $(\mathsf{Prove}_{\mathcal{R}}, \mathsf{Verify}_{\mathcal{R}})$:

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

It satisfies completeness, zero-knowledge, and standard soundness/proof of knowledge properties. Please refer to Appendix A for their formal definitions.

3 Technical Tools

3.1 Verifiable Non-Interactive Multiplication

We define the syntax of verifiable non-interactive multiplication (VNIM) as follows. While the objective of the original NIM [9] is to split the product $x_A \cdot x_B$ into additive shares s_A and s_B, our VNIM imposes additional requirements. Specifically, each party $\rho \in \{A,B\}$ must contribute a correct private input associated with a public value X_ρ (e.g., $X_\rho = x_\rho G$), and must publish the public share S_ρ corresponding to s_ρ (e.g., $S_\rho = s_\rho G$, for future use in a larger system). Zero-knowledge proofs are used throughout this process to ensure the correctness and consistency of both inputs and outputs.

Definitions. We present its formal definition as below.

Definition 3 (VNIM). *We assume that party ρ holds a private input x_ρ, while the corresponding public value X_ρ is publicly available. A Verifiable Non-Interactive Multiplication (VNIM) scheme* $\mathsf{VNIM} = (\mathsf{Setup}, \mathsf{VEncode}_\rho, \mathsf{VerifyInput}, \mathsf{VDecode}_\rho, \mathsf{VerifyOutput})_{\rho \in \{A,B\}}$ *has the following syntax:*

- $\mathsf{Setup}(1^\lambda, q) \to \mathsf{crs}$*: On input the security parameter λ and the prime q, the setup algorithm outputs a common reference string* crs.
- $\mathsf{VEncode}_\rho(\mathsf{crs}, x_\rho, X_\rho) \to (\mathsf{ec}_\rho, \pi_\rho, \mathsf{st}_\rho)$*: On input* crs*, private input x_ρ and its corresponding public value X_ρ, this verifiable encoding algorithm outputs the encoding* ec_ρ *of x_ρ, a consistency proof π_ρ guaranteeing that both* ec_ρ *and X_ρ contain the same x_ρ, and a private state* st_ρ*. Note that $(\mathsf{ec}_\rho, \pi_\rho)$ will be public.*
- $\mathsf{VerifyInput}(\mathsf{crs}, X_\rho, \mathsf{ec}_\rho, \pi_\rho) \to \{0,1\}$*: The verification algorithm outputs* 1 *if the proof π_ρ is valid, confirming that* ec_ρ *is a valid encoding of the correct private input corresponding to X_ρ; otherwise, it outputs* 0.
- $\mathsf{VDecode}_\rho(\mathsf{crs}, \mathsf{ec}_{\bar{\rho}}, \pi_{\bar{\rho}}, \mathsf{st}_\rho, x_\rho) \to (s_\rho, S_\rho, \tau_\rho)/\perp$*: Taking the counterparty's encoding* $\mathsf{ec}_{\bar{\rho}}$*, the private state* st_ρ*, and the input x_ρ as inputs, this verifiable decoding algorithm first invokes* $\mathsf{VerifyInput}$ *on $\pi_{\bar{\rho}}$. If it outputs 1, output a secret share s_ρ, its corresponding public value S_ρ, and a proof of correctness τ_ρ; otherwise, output $\perp$. Note that (S_ρ, τ_ρ) will be public.*
- $\mathsf{VerifyOutput}(\mathsf{crs}, \mathsf{ec}_A, \mathsf{ec}_B, S_\rho, \tau_\rho) \to \{0,1\}$*: The verification algorithm outputs* 1 *if the proof τ_ρ is valid, confirming that the public output S_ρ is well-formed with respect to its underlying share s_ρ, which is correctly decoded from the encodings* ec_A *and* ec_B*; otherwise, it outputs* 0.

Security Properties. In contrast to the original NIM's security model, we modify the privacy property to demand that both the encoding ec_ρ and the public output S_ρ leak no information about the private input x_ρ and the secret share s_ρ. Furthermore, we add a formal definition of verifiability, ensuring that ρ must use the correct input x_ρ and output the correct S_ρ.

Correctness: There exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\rho \in \{A, B\}$, and $x_\rho \in \mathbb{Z}_q$, we have that

$$\Pr\left[\begin{array}{l|l} & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, q) \\ s_A + s_B = x_A x_B & (\mathsf{ec}_\rho, \pi_\rho, \mathsf{st}_\rho) \leftarrow \\ 1 = \mathsf{VerifyInput}(\mathsf{ST}_\rho) & \mathsf{VEncode}_\rho(\mathsf{crs}, x_\rho, X_\rho) \\ 1 = \mathsf{VerifyOutput}(\mathsf{ST}'_\rho) & (s_\rho, S_\rho, \tau_\rho) \leftarrow \\ & \mathsf{VDecode}_\rho(\mathsf{crs}, \mathsf{ec}_{\bar\rho}, \pi_{\bar\rho}, \mathsf{st}_\rho, x_\rho) \end{array}\right] = 1 - \mathsf{negl}(\lambda).$$

with $\mathsf{ST}_\rho \coloneqq (\mathsf{crs}, \mathsf{ec}_\rho, X_\rho, \pi_\rho)$ and $\mathsf{ST}'_\rho \coloneqq (\mathsf{crs}, \mathsf{ec}_A, \mathsf{ec}_B, S_\rho, \tau_\rho)$.

Privacy: For all PPT adversary $\mathcal{A} \coloneqq (\mathcal{A}_1, \mathcal{A}_2)$, there exists a PPT simulator $\mathsf{Sim}_\rho \coloneqq (\mathsf{SimInput}_\rho, \mathsf{SimOutput}_\rho)$ and a negligible function $\mathsf{negl}(\cdot)$ such that for $\forall\ \rho \in \{A, B\}, \lambda \in \mathbb{N}, x_\rho \in \mathbb{Z}_q$, we have that

$$\left|\Pr\left[b^* = b \;\middle|\; \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, q) \\ (\mathsf{ec}_\rho^{(0)}, \pi_\rho^{(0)}, \mathsf{st}_\rho) \leftarrow \mathsf{VEncode}_\rho(\mathsf{crs}, x_\rho, X_\rho) \\ (\mathsf{ec}_{\bar\rho}, \pi_{\bar\rho}, \mathsf{st}_\mathcal{A}) \leftarrow \mathcal{A}_1(\mathsf{crs}) \\ (s_\rho, S_\rho^{(0)}, \tau_\rho^{(0)}) \leftarrow \mathsf{VDecode}_\rho(\mathsf{crs}, \mathsf{ec}_{\bar\rho}, \pi_{\bar\rho}, \mathsf{st}_\rho, x_\rho) \\ (\mathsf{ec}_\rho^{(1)}, \pi_\rho^{(1)}, \mathsf{st}_\rho^*) \leftarrow \mathsf{SimInput}_\rho(\mathsf{crs}, X_\rho) \\ (S_\rho^{(1)}, \tau_\rho^{(1)}) \leftarrow \mathsf{SimOutput}_\rho(\mathsf{crs}, \mathsf{ec}_{\bar\rho}, \pi_{\bar\rho}, \mathsf{st}_\rho^*, X_\rho) \\ b \leftarrow\!\$\ \{0,1\} \\ b^* \leftarrow \mathcal{A}_2(\mathsf{st}_\mathcal{A}, \mathsf{ec}_\rho^{(b)}, \pi_\rho^{(b)}, S_\rho^{(b)}, \tau_\rho^{(b)}) \end{array}\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda).$$

This property implies that even if the adversary acts as the counterparty $\bar\rho$ of role ρ, it cannot learn ρ's secrets x_ρ, s_ρ from the transmitted messages. This is guaranteed by the existence of a PPT simulator Sim_ρ that can simulate indistinguishable $(\mathsf{ec}_\rho, \pi_\rho, S_\rho, \tau_\rho)$ without taking x_ρ, s_ρ as input.

Verifiability: For all PPT adversary $\mathcal{A} \coloneqq (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for $\forall\ \rho \in \{A, B\}, \lambda \in \mathbb{N}, x_{\bar\rho} \in \mathbb{Z}_q$, we have that the advantage of $\mathcal{A}$ winning in the game $\mathsf{Exp}^\rho_{\mathcal{A},\mathsf{VNIM}}$ is negligible with $\mathsf{Adv}_{\mathsf{Exp}^\rho_{\mathcal{A},\mathsf{VNIM}}}(\lambda) \leq \mathsf{negl}(\lambda)$. The game $\mathsf{Exp}^\rho_{\mathcal{A},\mathsf{VNIM}}$ is defined as follows.

1. The challenger runs $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, q)$ and acts as the roles of $\bar\rho$ with the input $x_{\bar\rho} \in \mathbb{Z}_q$. The challenger runs $(\mathsf{ec}_{\bar\rho}, \pi_{\bar\rho}, \mathsf{st}_{\bar\rho}) \leftarrow \mathsf{VEncode}_{\bar\rho}(\mathsf{crs}, x_{\bar\rho}, X_{\bar\rho})$ and sends $(\mathsf{ec}_{\bar\rho}, \pi_{\bar\rho})$ to $\mathcal{A}_1$.
2. $\mathcal{A}_1$ (as the role ρ) generates $(X_\rho, \mathsf{ec}_\rho, \pi_\rho)$, sends it to the challenger and a state $\mathsf{st}_\mathcal{A}$ to $\mathcal{A}_2$. If $\mathsf{VerifyInput}(\mathsf{crs}, \mathsf{ec}_\rho, X_\rho, \pi_\rho) = 0$, the challenger aborts the game. Otherwise, it runs $(s_{\bar\rho}, S_{\bar\rho}, \tau_{\bar\rho}) \leftarrow \mathsf{VDecode}_{\bar\rho}(\mathsf{crs}, \mathsf{ec}_\rho, \pi_\rho, \mathsf{st}_{\bar\rho}, x_{\bar\rho})$ and sends $(S_{\bar\rho}, \tau_{\bar\rho})$ to $\mathcal{A}_2$.
3. With the input of $\mathsf{st}_\mathcal{A}$ and messages received from the challenger, $\mathcal{A}_2$ generates (S_ρ, τ_ρ) and sends it to the challenger. If $\mathsf{VerifyOutput}(\mathsf{crs}, \mathsf{ec}_A, \mathsf{ec}_B, S_\rho, \tau_\rho) = 0$, abort the game. Otherwise, the challenger runs an extractor $\mathcal{E}_1$ and an extractor $\mathcal{E}_2$ to get $\mathsf{witness}_1 \leftarrow \mathcal{E}_1^{\mathcal{A}_1}(\mathsf{crs}, \mathsf{ec}_\rho, X_\rho, \pi_\rho)$ and $\mathsf{witness}_2 \leftarrow \mathcal{E}_2^{\mathcal{A}_2}(\mathsf{crs}, S_\rho, \tau_\rho)$, where $\mathsf{witness}_1$ contains an x_ρ and $\mathsf{witness}_2$ contains an s_ρ.
4. The adversary wins if any of the following conditions happen:
   (a) $(\mathsf{witness}_1, X_\rho) \notin \mathcal{R}_1$ where $\mathcal{R}_1$ is the relation between the private input x_ρ and the public value X_ρ;
   (b) $(\mathsf{witness}_2, S_\rho) \notin \mathcal{R}_2$ where $\mathcal{R}_2$ is the relation between the private output s_ρ and the public value S_ρ;
   (c) $x_{\bar\rho} \cdot x_\rho \neq s_{\bar\rho} + s_\rho$.

Remark 1. For the generality, we require the existence of the second extractor $\mathcal{E}_2$ to extract s_ρ and check whether the adversary wins. In concrete constructions, this requirement can be relaxed if the S_ρ is generated by s_ρ via a one-way function $\mathsf{OWF}(\cdot)$ (e.g., $S_\rho = s_\rho G$ or $S_\rho = Hash(s_\rho)$). The challenger can compute $s_\rho = x_{\bar\rho} \cdot x_\rho - s_{\bar\rho}$ from its own $x_{\bar\rho}, s_{\bar\rho}$ and the x_ρ extracted by $\mathcal{E}_1$. Then it computes $S'_\rho = \mathsf{OWF}(s_\rho)$ and the adversary wins if $S'_\rho \neq S_\rho$.

Concrete Construction. In this section, we provide a class-group-based construction. While Abram et al. [1,2] showed that the original NIM can be instantiated over the Paillier group $\mathbb{Z}^*_{N^2}$, the RSA group $\mathbb{Z}^*_N$, or class groups, we opt for class groups due to their flexible plaintext space that

can natively align with the modulus q. Our VNIM framework can be generalized to Paillier and RSA groups as well, albeit at the cost of incorporating extra range proofs to handle the mismatch between their plaintext spaces and q.

Distributed Dlog (DDLog). Consider two parties, A and B, who hold related group elements. A holds $\mathfrak{f}^z\mathfrak{h}$ and B holds $\mathfrak{h}$, where $z \in \mathbb{Z}_q$ and $\mathfrak{h} \in \mathcal{G}^q$. The DDLog mechanism allows them to non-interactively derive additive shares $s_A, s_B \in \mathbb{Z}_q$ such that: $s_A + s_B = z \mod q$.

The DDLog mechanism is achieved using a *coset labeling function* ϕ that, for each coset $\mathcal{C}$ of $\mathcal{F}$, ϕ deterministically maps all elements in $\mathcal{C}$ to a unique representative $\mathfrak{h}^*$. More intuitively, for each $\mathfrak{h} \in \mathcal{G}^q$, let $\mathcal{C}_\mathfrak{h} := \{\mathfrak{f}^x\mathfrak{h} \mid x \in \mathbb{Z}_q\}$, and there exists $\mathfrak{h}^* \in \mathcal{C}_\mathfrak{h}$ such that for each $\mathfrak{a} \in \mathcal{C}_\mathfrak{h}$ we have $\phi(\mathfrak{a}) = \mathfrak{h}^*$.

Based on this, the shares are computed as follows: A computes $s_A = \mathsf{Dlog}(\mathfrak{f}^z\mathfrak{h}/\phi(\mathfrak{f}^z\mathfrak{h}))$; B computes $s_B = \mathsf{Dlog}(\phi(\mathfrak{h})/\mathfrak{h})$. We have $s_A + s_B = z \mod q$, as $\phi(\mathfrak{f}^z\mathfrak{h}) = \phi(\mathfrak{h}) \in \mathcal{C}_\mathfrak{h}$. The concrete mathematical details of the coset labeling function $\phi(\cdot)$ can be found in [1].

VNIM Construction. In this construction, party A's public value corresponds to a Pedersen commitment $X_A = x_A G + k_A H$, where $G, H \in \mathbb{G}$ and $k_A \in \mathbb{Z}_q$ is the randomness, whereas party B's public value is given by $X_B = x_B G$. The revealed public shares take the form $S_\rho = s_\rho G$ for each $\rho \in \{A, B\}$. In line with the original NIM framework, we utilize Pedersen commitments over $\mathcal{G}^q$ to form party A's encoding and CL encryption for party B's encoding. What sets our VNIM apart is the incorporation of suitable zero-knowledge proofs to enforce the validity of inputs and outputs. We define the following relations needed for our VNIM construction and show their concrete constructions in Appendix A.1.

$$\mathcal{R}_{\mathsf{ecA}} = \{(\mathsf{ec}_A, X_A), (x_A, k_A, r_A)) : X_A = x_A G + k_A H, \mathsf{ec}_A = \mathfrak{g}_0^{r_A}\mathfrak{g}_1^{x_A}\}.$$

$$\mathcal{R}_{\mathsf{ecB}} = \{((\mathsf{ec}_B, X_B), (x_B, r_B)) : X_B = x_B G, \mathsf{ec}_B = (\mathfrak{g}_0^{r_B}, \mathfrak{f}^{x_B}\mathfrak{g}_1^{r_B})\}.$$

$$\mathcal{R}_{\mathsf{outA}} = \left\{ \begin{array}{l|l} (S_A, \mathfrak{h}^*, \mathsf{ec}_A, \mathsf{ec}_B = (\mathfrak{c}_0, \mathfrak{c}_1)), & \mathfrak{h}^* = \mathfrak{c}_0^{r_A}\mathfrak{c}_1^{x_A}\mathfrak{f}^{-s_A} \\ (s_A, x_A, r_A) & \mathsf{ec}_A = \mathfrak{g}_0^{r_A}\mathfrak{g}_1^{x_A} \\ & S_A = s_A G \end{array} \right\}.$$

$$\mathcal{R}_{\mathsf{outB}} = \left\{ \begin{array}{l|l} (S_B, \mathfrak{h}^*, \mathsf{ec}_A, \mathfrak{c}_0), & \mathfrak{h}^* = \mathsf{ec}_A^{r_B}\mathfrak{f}^{s_B} \\ (s_B, r_B) & \mathfrak{c}_0 = \mathfrak{g}_0^{r_B} \\ & S_B = s_B G \end{array} \right\}.$$

The specific algorithmic framework is as follows.

- $\mathsf{Setup}(1^\lambda, q)$: Generates the public parameters $\mathsf{crs} := (\mathfrak{f}, \mathfrak{g}_0, \mathfrak{g}_1, G, H)$, where $\mathfrak{g}_0, \mathfrak{g}_1$ are randomly sampled generators of $\mathcal{G}_q$ and G, H are random generators of the EC group $\mathbb{G}$.
- $\mathsf{VEncode}_A(\mathsf{crs}, x_A, X_A)$: Sample a random value $r_A \leftarrow\!\!\$\ \mathcal{D}_q$. Compute the commitment $\mathsf{ec}_A := \mathfrak{g}_0^{r_A}\mathfrak{g}_1^{x_A}$ and define the internal state $\mathsf{st}_A := r_A$. Invoke the zero-knowledge proof
$$\pi_A \leftarrow \mathsf{Prove}_{\mathcal{R}_{\mathsf{ecA}}}((\mathsf{ec}_A, X_A), (x_A, k_A, \mathsf{st}_A)).$$
Output $(\mathsf{ec}_A, \pi_A, \mathsf{st}_A)$.
- $\mathsf{VEncode}_B(\mathsf{crs}, x_B, X_B)$: Sample a random value $r_B \leftarrow\!\!\$\ \mathcal{D}_q$. Compute the ciphertext $\mathsf{ec}_B := (\mathfrak{g}_0^{r_B}, \mathfrak{f}^{x_B}\mathfrak{g}_1^{r_B})$ and define the internal state $\mathsf{st}_B := r_B$. Invoke the zero-knowledge proof $\pi_B \leftarrow \mathsf{Prove}_{\mathcal{R}_{\mathsf{ecB}}}((\mathsf{ec}_B, X_B), (x_B, k_B, \mathsf{st}_B))$. Output $(\mathsf{ec}_B, \pi_B, \mathsf{st}_B)$.
- $\mathsf{VerifyInput}(\mathsf{crs}, \mathsf{ec}_\rho, X_\rho, \pi_\rho), \forall \rho \in \{A, B\}$: Invoke the verify algorithm $\mathsf{Verify}_{\mathcal{R}_{\mathsf{ec}_\rho}}((\mathsf{ec}_\rho, X_\rho), \pi_\rho) \to b$ and output b.
- $\mathsf{VDecode}_A(\mathsf{crs}, \mathsf{ec}_B, \pi_B, \mathsf{st}_A, x_A)$: If π_B is not valid, output $\perp$. Otherwise, parse $\mathsf{ec}_B = (\mathfrak{c}_0, \mathfrak{c}_1)$ and $\mathsf{st}_A = r_A$. Compute the element $\mathfrak{a} := \mathfrak{c}_0^{r_A}\mathfrak{c}_1^{x_A}$. Apply the labeling function to get $\mathfrak{h}^* := \phi(\mathfrak{a})$. Get the share $s_A := \mathsf{Dlog}(\mathfrak{a}/\mathfrak{h}^*)$ and compute $S_A := s_A G$. Invoke the zero-knowledge proof $\tau'_A \leftarrow \mathsf{Prove}_{\mathcal{R}_{\mathsf{outA}}}((S_A, \mathfrak{h}^*, \mathsf{ec}_A, \mathsf{ec}_B), (s_A, x_A, r_A))$. Output $(s_A, S_A, \tau_A := (\mathfrak{h}^*, \tau'_A))$.
- $\mathsf{VDecode}_B(\mathsf{crs}, \mathsf{ec}_A, \pi_A, \mathsf{st}_B, x_B)$: If π_A is not valid, output $\perp$. Otherwise, parse $\mathsf{ec}_A = \mathfrak{c}$ and $\mathsf{st}_B = r_B$. Compute the element $\mathfrak{b} := \mathfrak{c}^{r_B}$. Apply the labeling function to get $\mathfrak{h}^* := \phi(\mathfrak{b})$. Get the share $s_B := \mathsf{Dlog}(\mathfrak{h}^*/\mathfrak{b})$ and compute $S_B := s_B G$. With $\mathsf{ec}_B = (\mathfrak{c}_0, \mathfrak{c}_1)$, invoke the zero-knowledge proof $\tau'_B \leftarrow \mathsf{Prove}_{\mathcal{R}_{\mathsf{outB}}}((S_B, \mathfrak{h}^*, \mathsf{ec}_A, \mathfrak{c}_0), (s_B, r_A))$. Output $(s_B, S_B, \tau_B := (\mathfrak{h}^*, \tau'_B))$.
- $\mathsf{VerifyOutput}(\mathsf{crs}, \mathsf{ec}_A, \mathsf{ec}_B, S_\rho, \tau_\rho), \forall \rho \in \{A, B\}$: Parse $\tau_\rho = (\mathfrak{h}^*, \tau'_\rho)$. Output 1 if $\phi(\mathfrak{h}^*) = \mathfrak{h}^*$ and $\mathsf{Verify}_{\mathcal{R}_{\mathsf{out}_\rho}}((S_\rho, \mathfrak{h}^*, \mathsf{ec}_A, \mathsf{ec}_B), \tau'_\rho) = 1$; otherwise, output 0.

Lemma 2. *If $\mathcal{R}_{\mathsf{ecA}}$-relation, $\mathcal{R}_{\mathsf{ecB}}$-relation proofs are zero-knowledge proofs of knowledge*[5]*, and $\mathcal{R}_{\mathsf{outA}}$-relation, $\mathcal{R}_{\mathsf{outB}}$-relation proofs are zero-knowledge proofs with standard soundness, our VNIM construction satisfies correctness, privacy, and verifiability. (The proof is deferred to Appendix B.4.)*

Remark 2. If S_ρ is not required in the application, the $\mathsf{VDecode}_\rho$ algorithm reduces to $\mathsf{Decode}_\rho(\mathsf{crs}, \mathsf{ec}_{\bar{\rho}}, \mathsf{st}_\rho, x_\rho) \to s_\rho$, only outputting s_ρ.

Optimization in Multi-Party Setting. VNIM can be applied in multi-party scenarios, such as threshold signatures. Specifically, each party P_i holds additive shares x_i and y_i of x and y, respectively. VNIM decomposes each product $x_i y_j = \alpha_{ij} + \beta_{ji}$, where P_i obtains α_{ij} and P_j obtains β_{ji}. Consequently, each party P_i can compute its additive share of xy as $\sum_j(\alpha_{ij} + \beta_{ij})$. However, if P_i needs to publicly output $\alpha_{ij}G$ or $\beta_{ij}G$ for each j (for robustness), it must prove the correctness of each term during the execution of $\mathsf{VDecode}$.

Here, we introduce an optimization to aggregate these proofs into a single proof. Given that P_i has received $\mathsf{ec}_{Ax_j}, \mathsf{ec}_{By_j}$ for each j, we take $\mathsf{VDecode}_B$ as an example to demonstrate how P_i can aggregate m executions of $\{\mathsf{VDecode}_B(\mathsf{crs}, \mathsf{ec}_{Ax_j}, \mathsf{st}_{By_i}, y_i)\}_{j\neq i}$ into a single procedure $\mathsf{VDecode}_B^m$ as below.

- $\mathsf{VDecode}_B^m(\mathsf{crs}, \{\mathsf{ec}_{Ax_j}\}_{j\neq i}, \mathsf{st}_{By_i}, y_i) \to (\{\beta_{ij}, B_{ij}\}_{j\neq i}, \tau_i)$: Parse $\mathsf{ec}_{Ax_j} = \mathfrak{c}_j,\ \forall j \neq i$, $\mathsf{st}_{By_i} = r_i$. Run $\mathsf{Decode}_B(\mathsf{crs}, \mathsf{ec}_{Ax_j}, \mathsf{st}_{By_i}, y_i) \to \beta_{ij}$ and compute $B_{ij} = \beta_{ij}G$ for each $j \neq i$. Generate coefficients $(e_1, \cdots, e_m) \leftarrow \mathsf{H}(\{\mathfrak{c}_j, B_{ij}, \mathfrak{h}_{ij}^*\}_{j\neq i}, \mathfrak{c}_0)$ where H is a random oracle, $\mathfrak{h}_{ij}^* = \phi(\mathfrak{c}_j^{r_i}) = \mathfrak{c}_j^{r_i}\mathfrak{f}^{\beta_{ij}}$, $\mathfrak{c}_0 = \mathfrak{g}_0^{r_i}$, and each $e_j \in [0, 2^\lambda]$.
  1. Assemble the statement: $\mathfrak{c} = \prod_j \mathfrak{c}_j^{e_j}$, $B_i = \sum_j e_j B_{ij}$, $\mathfrak{h}_i^* = \prod_j (\mathfrak{h}_{ij}^*)^{e_j}$
  2. Assemble the witness: $\beta_i = \sum_j e_j \beta_{ij} \bmod q$
  3. Invoke the zero-knowledge proof $\tau' \leftarrow \mathsf{Prove}_{\mathcal{R}_{\mathsf{outB}}}((B_i, \mathfrak{h}_i^*, \mathfrak{c}, \mathfrak{c}_0), (\beta_i, r_i))$
  4. $\tau_i \coloneqq (\{\mathfrak{h}_{ij}^*\}_{j\neq i}, \tau'\})$, output $(\{\beta_{ij}, B_{ij}\}_{j\neq i}, \tau_i)$.
- $\mathsf{VerifyOutput}^m(\mathsf{crs}, \{\mathsf{ec}_{Ax_j}\}_{j\neq i}, \mathsf{ec}_{By_i}, \{B_{ij}\}_{j\neq i}, \tau_i) \to b$: Assemble the statement:
  1. Parse $\mathsf{ec}_{Ax_j} = \mathfrak{c}_j,\ \forall j \neq i$, $\mathsf{ec}_{By_i} = (\mathfrak{c}_0, \mathfrak{c}_1)$, $\tau_i = (\{\mathfrak{h}_{ij}^*\}_{j\neq i}, \tau'\}$
  2. $(e_1, \cdots, e_m) \leftarrow \mathsf{H}(\{\mathfrak{c}_j, B_{ij}, \mathfrak{h}_{ij}^*\}_{j\neq i}, \mathfrak{c}_0)$
  3. $\mathfrak{c} = \prod_j \mathfrak{c}_j^{e_j}$, $B_i = \sum_j e_j B_{ij}$, $\mathfrak{h}_i^* = \prod_j (\mathfrak{h}_{ij}^*)^{e_j}$

  Output 1 if $\phi(\mathfrak{h}_{ij}^*) = \mathfrak{h}_{ij}^*,\ \forall j \neq i$ and $\mathsf{Verify}_{\mathcal{R}_{\mathsf{outB}}}((B_i, \mathfrak{h}_i^*, \mathfrak{c}, \mathfrak{c}_0), \tau') = 1$; otherwise, output 0.

Similarly, P_i can aggregate m executions of $\{\mathsf{VDecode}_A(\mathsf{crs}, \mathsf{ec}_{By_j}, \mathsf{st}_{Ax_i}, x_i)\}_{j\neq i}$ into a single procedure, denoted as $\mathsf{VDecode}_A^m$. As the technical details are analogous, we omit them here for brevity.

3.2 Special PVSS scheme PVSSeq

We present a special PVSS scheme PVSSeq, where the dealer shares a polynomial $f(\cdot)$ accompanied by a proof that $f(\cdot)$ satisfies

1. the degree of $f(\cdot)$ is no larger than $t + \ell - 1$;
2. $f(0) = f(-1) = \cdots = f(-\ell + 1)$.

The syntax of PVSSeq is similar to that of the standard PVSS in Definition 2, except that the Share and $\mathsf{VerifySharing}$ algorithms additionally ensure that Condition 2 holds.

Intuitively, we let the dealer not only broadcast encrypted shares but also the Pedersen commitments of the shares. Verifying Condition 2 over Pedersen commitments would be significantly more efficient than over encrypted shares, since EC operations are 3 orders of magnitude faster than class group operations. Specifically, it can be verified via $\mathsf{Com}(f(0)) = \cdots = \mathsf{Com}(f(-\ell+1))$, where the verifier can compute each $\mathsf{Com}(f(i))$ from the Pedersen commitments of the shares via Lagrange interpolation. At the same time, we translate the dual-code technique from [13] to the Pedersen commitment. It is used to ensure that each party's share is derived from the same polynomial $f(\cdot)$ of degree at most $t + \ell - 1$ (i.e., Condition 1).

[5] We only require a proof of partial knowledge, specifically for x_A and x_B.

Another key part of our approach is proving the consistency between each encrypted share and its corresponding committed share, which is the relation

$$\mathcal{R}_{\mathsf{share}} = \{(\{ek_j, \mathcal{P}_j, \mathcal{C}_j\}_{j\in S}, (\{a_j, a'_j\}_{j\in S}, \rho)) : \\ \forall\ j \in S,\ \mathcal{C}_j = \mathsf{CL.Enc}(ek_j, a_j, \rho), \mathcal{P}_j = a_j G + a'_j H\}.$$

We defer their concrete constructions to the Appendix A.1, which is aggregated into a single constant-size proof. We also require the zero-knowledge proof of knowledge for the relation $\mathcal{R}_{\mathsf{key}} = \{(ek, dk) : ek = \mathfrak{g}_q^{dk}\}$.

Our PVSSeq scheme consists of the tuple of algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Share}, \mathsf{VerifySharing})$.

- $\mathsf{PVSSeq.Setup}(1^\lambda, q, \ell, t, n) \to \mathsf{pp}$: let $\mathsf{pp}_{\mathsf{Sh}} := (q, \ell, t, n, G, H)$ where G, H are two independent and random generators of EC group $\mathbb{G}$, run $\mathsf{pp}_{\mathsf{CL}} := (q, \tilde{s}, \mathfrak{f}, \mathfrak{g}_q, \mathcal{F}, \mathcal{G}) \leftarrow \mathsf{CLGen}(1^\lambda, q)$, and output $\mathsf{pp} := (\mathsf{pp}_{\mathsf{Sh}}, \mathsf{pp}_{\mathsf{CL}})$
- $\mathsf{PVSSeq.KeyGen}(\mathsf{pp}_{\mathsf{CL}}, i) \to (ek_i, dk_i, \pi_{\mathsf{key}_i})$: generate $(ek_i, dk_i) \leftarrow \mathsf{CL.KGen}(\mathsf{pp}_{\mathsf{CL}})$ and the proof $\pi_{\mathsf{key}_i} \leftarrow \mathsf{Prove}_{\mathcal{R}_{\mathsf{key}}}(ek_i, dk_i)$. Output $(ek_i, dk_i, \pi_{\mathsf{key}_i})$.
- $\mathsf{PVSSeq.Share}(\mathsf{pp}, \{ek_i\}_{i\in[n]}, x) \to (\{\mathcal{P}_j, \mathcal{C}_j\}_j, \pi_{\mathsf{share}})$:
  1. Create a packed sharing of x: pick two degree-$(t+\ell-1)$ polynomials $f(\cdot), f'(\cdot)$ s.t. $f(-\nu+1) = x, f'(-\nu+1) = r, \forall \nu \in [\ell]$ with $r \leftarrow\!\!\$\ \mathbb{Z}_q$ and set $m_j = f(j), m'_j = f'(j), \forall j \in [n]$.
  2. Generate Pedersen commitments of shares: $\mathcal{P}_j := m_j G + m'_j H \in \mathbb{G}, \forall j \in [n]$
  3. Generate CL ciphertexts of shares: $\mathcal{C}_j := (\mathcal{R}, ek_j^\rho \mathfrak{f}^{m_j}), \forall j \in [n]$ with $\mathcal{R} = \mathfrak{g}_q^\rho$ and $\rho \leftarrow\!\!\$\ \mathcal{D}_q$.
  4. Generate the proof $\pi_{\mathsf{share}} \leftarrow \mathsf{Prove}_{\mathcal{R}_{\mathsf{share}}}((\{\mathcal{P}_j, \mathcal{C}_j\}_j), (\{m_j, m'_j\}_j, \rho))$.
  5. Output $(\{\mathcal{P}_j, \mathcal{C}_j\}_{j\in[n]}, \pi_{\mathsf{share}})$.
- $\mathsf{PVSSeq.VerifySharing}(\mathsf{pp}, \{ek_j, \mathcal{P}_j, \mathcal{C}_j\}_{j\in[n]}, \pi_{\mathsf{share}}) \to b$:
  1. Check the equality of packed secrets:

$$\sum_{j\in[n]} \lambda_j^{(1)} \mathcal{P}_j \stackrel{?}{=} \sum_{j\in[n]} \lambda_j^{(2)} \mathcal{P}_j \stackrel{?}{=} \cdots \stackrel{?}{=} \sum_{j\in[n]} \lambda_j^{(\ell)} \mathcal{P}_j.$$

  2. Check the degree of shared polynomial via the dual-code technique (c.f. Appendix D): with a random degree-$(n-t-\ell-1)$ polynomial $p(\cdot)$ and $v_j = \prod_{\mu\in[n]\setminus\{j\}} (\mu - j)^{-1} \in \mathbb{Z}_q$, $\sum_{j\in[n]} p(j) v_j \mathcal{P}_j \stackrel{?}{=} O$, where $O \in \mathbb{G}$ is the identity element in $\mathbb{G}$ with $\mathsf{Dlog}(O) = 0$.
  3. Run $\mathsf{Verify}_{\mathcal{R}_{\mathsf{share}}}$ on the proof π_{share} with respect to statement $\{ek_j, \mathcal{P}_j, \mathcal{C}_j\}_{j\in[n]}$.
  4. Output $b = 1$ if all three verifications pass, 0 otherwise.

Lemma 3. *If $\mathcal{R}_{\mathsf{key}}$-relation proof is a zero-knowledge proof of knowledge, and $\mathcal{R}_{\mathsf{share}}$-relation proof is a zero-knowledge proof with standard soundness, the* PVSSeq *scheme satisfies correctness, verifiability, and privacy properties (defined in Appendix C).*

Compared to the PVSS construction in [14], our PVSSeq additionally requires publishing the commitments to the shares and proving that the committed values correspond to the encrypted shares. Therefore, the scheme's privacy is still guaranteed by the zero-knowledge property of π_{share} and the hiding property of Pedersen commitments, while its verifiability is ensured by standard soundness.

4 Batched Robust Threshold ECDSA

4.1 Distributed Key Generation

In this phase, the parties aim to jointly generate a public key $X = xG$, and each party receives a t-out-of-n share x_i of x. Specifically, we need a packed secret sharing for identical secrets $[x, \cdots, x] \in \mathbb{Z}_q^\ell$. Intuitively, each party $P_i,\ i \in [n]$ invokes PVSSeq to share a uniformly random secret value. Then each party will get a t-out-of-n share x_i and output the public share $X_i = x_i G$. At the same time, each party invokes $\mathsf{VEncode}_B$ to generate x_i's encoding.

We provide our concrete construction of the distributed key generation protocol in Figure 2. Specifically, to guarantee public key share X_i's correctness, a $\mathcal{R}_{\mathsf{DL\text{-}D}}$-relation zero-knowledge proof is required, with $\mathcal{R}_{\mathsf{DL\text{-}D}} = \{((\mathcal{C}_a, ek, A), (a, dk)) : A = aG, a = \mathsf{CL.Dec}(dk, \mathcal{C}_a), ek = \mathfrak{g}_q^{dk}\}$. The concrete construction has been shown in Appendix A.1.

Fig. 1: Offline Phase of Signing Protocol.

Generate $\ell\hat{\ell}$ distinct R's.

- Round 1. Each party P_i (as a dealer) invokes PVSS: Pick $\mathbf{s}_i \leftarrow\!\!\$\ \mathbb{Z}_q^\ell$ and run $(\{\mathcal{C}_{ij}\}_{j\in S}, \pi_{\mathsf{Sh}_i}) \leftarrow \mathsf{PVSS.Share}(\mathsf{pp}, \{ek_j\}_j, \mathbf{s}_i)$. Broadcast $(\{\mathcal{C}_{ij}\}_{j\in S}, \pi_{\mathsf{Sh}_i})$.
- Round 2. Now, S denotes the set of i for which $\mathsf{PVSS.VerifySharing}(\mathsf{pp}, \{ek_j, \mathcal{C}_{ij}\}_{j\in S}, \pi_{\mathsf{Sh}_i}) \to 1$. Each party $P_i, i \in S$ combines the shares of $\ell\hat{\ell}$ distinct values of k:
  1. Parse $\mathcal{C}_{ji} = (\mathcal{R}_j, \mathcal{B}_{ji}), \forall j \in S$ and decrypt it $k_{ji} = \mathsf{Dlog}(\mathcal{B}_{ji}/\mathcal{R}_j^{dk_i})$. With the $\hat{\ell} \times \hat{n}$-dimension SIM $\boldsymbol{\Psi} = [\psi_j^{(\mu)}]_{j\in[\hat{n}]}^{\mu\in[\hat{\ell}]}$, generate $k_i^{(\mu)} = \sum_{j\in S} \psi_j^{(\mu)} k_{ji}$ and $R_i^{(\mu)} = k_i^{(\mu)} G \in \mathbb{G}$..
  2. Generate an aggregated proof $\tau_{\mathsf{dld}_i} \leftarrow \mathsf{Prove}_{\mathcal{R}_{\mathsf{DL\text{-}D}}^m}(\{\mathcal{R}^{(\mu)}, \mathcal{B}_i^{(\mu)}, R_i^{(\mu)}\}_{\mu\in[\hat{\ell}]}, (\{k_i^{(\mu)}\}_{\mu\in[\hat{\ell}]}, dk_i))$, with $\mathcal{R}^{(\mu)} = \prod_{j\in S} \mathcal{R}_j^{\psi_j^{(\mu)}}$ and $\mathcal{B}_i^{(\mu)} = \prod_{j\in S} \mathcal{B}_{ji}^{\psi_j^{(\mu)}}$. Broadcast $(\{R_i^{(\mu)}\}_{\mu\in[\hat{\ell}]}, \tau_{\mathsf{dld}_i})$.
- Output. Now, S is the set of i s.t. τ_{dld_i} is valid. If $|S| \geq t+\ell$, output $R^{(\mu\nu)} = \sum_{i\in S} \lambda_i^{(\nu)} R_i^{(\mu)}, \forall \mu \in [\hat{\ell}], \nu \in [\ell]$.

Calls of VNIM. Each party P_i holds the private inputs $\left(x_i, \{k_i^{(\mu)}\}_{\mu\in[\hat{\ell}]}\right)$. For each $\mu \in [\hat{\ell}]$, let $k_i = k_i^{(\mu)}$, $R_i = R_i^{(\mu)}$.

- Round 1. Each party picks $\gamma_i \leftarrow\!\!\$\ \mathbb{Z}_q$ and computes $\Gamma_i = \gamma_i G + r_i H$ with $r_i \leftarrow\!\!\$\ \mathbb{Z}_q$. Then it runs $(\mathsf{ec}_{A\gamma_i}, \pi_{A\gamma_i}, \mathsf{st}_{A\gamma_i}) \leftarrow \mathsf{VEncode}_A(\mathsf{crs}, \gamma_i, \Gamma_i)$ as a role of A. Broadcast $(\Gamma_i, \mathsf{ec}_{A\gamma_i}, \pi_{A\gamma_i})$.
- Round 2. If $\mathsf{VerifyInput}(\mathsf{crs}, \mathsf{ec}_{A\gamma_j}, \Gamma_j, \pi_{A\gamma_j}) \to 0$, set $S = S \setminus \{j\}$. As a role of B, each party P_i runs $(\mathsf{ec}_{Bk_i}, \pi_{Bk_i}, \mathsf{st}_{Bk_i}) \leftarrow \mathsf{VEncode}_B(\mathsf{crs}, k_i, R_i)$, $(\{\beta_{ij}, B_{ij}\}_{j\neq i}, \tau_i) \leftarrow \mathsf{VDecode}_B^m(\mathsf{crs}, \{\mathsf{ec}_{A\gamma_j}\}_{j\neq i}, \mathsf{st}_{Bk_i}, k_i)$, and $(\{\hat{\beta}_{ij}, \hat{B}_{ij}\}_{j\neq i}, \hat{\tau}_i) \leftarrow \mathsf{VDecode}_B^m(\mathsf{crs}, \{\mathsf{ec}_{A\gamma_j}\}_{j\neq i}, \mathsf{st}_{Bx_i}, x_i)$. Party P_i broadcasts $(\mathsf{ec}_{Bk_i}, \pi_{Bk_i}, \{B_{ij}, \hat{B}_{ij}\}_{j\neq i}, \tau_i, \hat{\tau}_i)$.
- Output. If $\mathsf{VerifyInput}(\mathsf{crs}, \mathsf{ec}_{Bk_j}, R_j, \pi_{Bk_j}) \to 0$ or $\mathsf{VerifyOutput}^m(\mathsf{crs}, \{\mathsf{ec}_{A\gamma_t}\}_{t\neq j}, \mathsf{ec}_{Bk_j}, \{B_{jt}\}_{t\neq j}, \tau_j) \to 0$ or $\mathsf{VerifyOutput}^m(\mathsf{crs}, \{\mathsf{ec}_{A\gamma_t}\}_{t\neq j}, \mathsf{ec}_{Bx_j}, \{\hat{B}_{jt}\}_{t\neq j}, \hat{\tau}_j) \to 0$, set $S = S \setminus \{j\}$. As a role of A, each party P_i runs $\alpha_{ij} \leftarrow \mathsf{Decode}_A(\mathsf{crs}, \mathsf{ec}_{Bk_j}, \mathsf{st}_{A\gamma_i}, \gamma_i)$ and $\hat{\alpha}_{ij} \leftarrow \mathsf{Decode}_A(\mathsf{crs}, \mathsf{ec}_{Bx_j}, \mathsf{st}_{A\gamma_i}, \gamma_i)$. Compute $\delta_{ij} = \alpha_{ij} + \beta_{ij} \bmod q, \hat{\delta}_{ij} = \hat{\alpha}_{ij} + \hat{\beta}_{ij} \bmod q$ for each $j \in S \setminus \{i\}$.

4.2 Threshold Signing

Let $S \subseteq Q$ be the set of signers' indices, where $Q \subseteq [n]$ denotes the set of parties who have honestly executed the DKG protocol. It is robust if $|S| = \hat{n} \geq 2t + \ell$, where ℓ is the packed size of packed secret sharing. For the sake of simplicity and clarity, we treat the set S as $[\hat{n}]$. Let another packed parameter corresponding to the SIM technique be $\hat{\ell} = \hat{n} - t$.

Offline phase It is divided into two phases: (1) the batch generation of $\hat{\ell}\ell$ random EC points R for deriving $\hat{\ell}\ell$ r-values, and (2) the invocation of VNIMs to process the multiplications in the s-component. Both phases require two rounds and can be run together, thus the offline phase requires two rounds in total. We provide the formal description in Figure 1.

Generate $\ell\hat{\ell}$ distinct R's. The objective here is for the parties to jointly generate $R = kG$, where each party holds a t-threshold share k_i of the secret k. Packed secret sharing can embed ℓ values of k within a single polynomial. Consequently, each party invoking packed PVSS can naturally generate ℓ values of R. Further, our design incorporates a super-invertible matrix (SIM) technique to increase the batch size to $\ell\hat{\ell}$, provided there are at least $\hat{\ell} = \hat{n} - t$ honest signing parties. Specifically, we select an $\hat{\ell} \times \hat{n}$ Vandermonde matrix as a SIM $\boldsymbol{\Psi}$. Each signer takes the $\hat{n}$ received shares (of which at least $\hat{\ell}$ are controlled by honest parties) and utilizes $\boldsymbol{\Psi}$ to combine them into $\hat{\ell}$ independent and random shares, $[k_i^{(1)}, \cdots, k_i^{(\hat{\ell})}]^\mathsf{T} = \boldsymbol{\Psi}[k_{1i}, \cdots, k_{\hat{n}i}]^\mathsf{T}$[6]. It means that, although each signer shares only a single polynomial in the first round, $\hat{\ell}$ polynomials can be assembled. Furthermore, because these polynomials are ℓ-packed, the overhead of the first round is effectively amortized across $\ell\hat{\ell}$ values of R. During this phase, we require an aggregatable zero-knowledge proof

[6] If a malicious party P_j fails to provide honest first-round messages, the value of k_{ji} can be set to zero.

Fig. 2: Distributed Key Generation Protocol.

Setup. On input the security parameter λ, parties run $\mathsf{pp} \leftarrow \mathsf{PVSSeq.Setup}(1^\lambda, q, \ell, t, n)$, $\mathsf{crs} \leftarrow \mathsf{VNIM.Setup}(1^\lambda, q)$, $\mathsf{PVSSeq.KeyGen}(\mathsf{pp}_{\mathsf{CL}}, i) \rightarrow (ek_i, dk_i, \pi_{\mathsf{key}_i})$. Let Q be the set of i of which π_{key_i} is valid.

Round 1. Each party P_i (as a dealer) picks $w_i \leftarrow\!\!\$\ \mathbb{Z}_q$, runs $(\{\mathcal{P}_{ij}, \mathcal{C}_{ij}\}_{j\in[n]}, \pi_{\mathsf{share}_i}) \leftarrow \mathsf{PVSSeq.Share}(\mathsf{pp}, \{ek_j\}_j, w_i)$. Broadcast $(\{\mathcal{P}_{ij}, \mathcal{C}_{ij}\}_{j\in[n]}, \pi_{\mathsf{share}_i})$.

Round 2. If $\mathsf{PVSSeq.VerifySharing}(\mathsf{pp}, \{ek_j, \mathcal{P}_{ij}, \mathcal{C}_{ij}\}_{j\in[n]}, \pi_{\mathsf{share}_i}) \rightarrow 0$, set $Q = Q \setminus \{j\}$. Each party $P_i, i \in Q$ does:

1. Parse $\mathcal{C}_{ji} = (\mathcal{R}_j, \mathcal{B}_{ji}), \forall j \in Q$. Assemble the ciphertext $\mathcal{R} = \prod_{j\in Q} \mathcal{R}_j, \mathcal{B}_i = \prod_{j\in Q} \mathcal{B}_{ji}$ and decrypt it $x_i = \mathsf{Dlog}(\mathcal{B}_i / \mathcal{R}^{dk_i})$
2. Generate $X_i = x_i G \in \mathbb{G}$ and the proof $\pi_{\mathsf{dld}_i} \leftarrow \mathsf{Prove}_{\mathcal{R}_{\mathsf{DL\text{-}D}}}((\mathcal{R}, \mathcal{B}_i, ek_i, X_i), (x_i, dk_i))$. Run $(\mathsf{ec}_{Bx_i}, \pi_{Bx_i}, \mathsf{st}_{Bx_i}) \leftarrow \mathsf{VEncode}_B(\mathsf{crs}, x_i, X_i)$. Broadcast $(X_i, \pi_{\mathsf{dld}_i}, \mathsf{ec}_{Bx_i}, \pi_{Bx_i})$[a].

Output. $X = \sum_{i\in Q} \lambda_i^{(1)} X_i$ if $|Q| \geq t + \ell$. Here, Q denotes the set of i s.t. π_{dld_i} and π_{Bx_i} are valid.

[a] Because both π_{dld_i} and π_{Bx_i} prove statements regarding x_i, certain components across the two proofs can be merged to optimize proof size and computation.

that supports multiple statements of relation $\mathcal{R}_{\mathsf{DL\text{-}D}}$, that is $\mathcal{R}^m_{\mathsf{DL\text{-}D}} = \{((\{\mathcal{C}_i, A_i\}_i, ek), (a_i, dk)) : A_i = a_i G, a_i = \mathsf{CL.Dec}(dk, \mathcal{C}_i), ek = \mathfrak{g}_q^{dk}\}$. The concrete constructions are deferred to Appendix A.1.

Calls of VNIM. It generates $\alpha_{ij}, \hat{\alpha}_{ij}$ for P_i and $\beta_{ji}, \hat{\beta}_{ji}$ for P_j s.t $\alpha_{ij} + \beta_{ji} = \gamma_i k_j$ and $\hat{\alpha}_{ij} + \hat{\beta}_{ji} = \gamma_i x_j, \forall i, j \in S$. For each $\mu \in [\hat{\ell}]$, $k_i = k_i^{(\mu)}$ must be masked by a corresponding γ_i. Furthermore, like k_i, the values of γ_i should also be ℓ-packed. Re-executing the entire previous process specifically for γ would undoubtedly be costly, as γ serves no purpose other than masking. Instead, we leverage the SIM concept again. In this instance, however, we utilize a Lagrange coefficient matrix (defined in Section 2.2) as the SIM. As demonstrated in Lemma 1, the Lagrange matrix possesses the super-invertible property. Thus, we have $\gamma^{(\nu)} = \sum_{j\in S} \lambda_j^{(\nu)} \gamma_j, \forall \nu \in [\ell]$ are ℓ independent and random nonces if $|S| \geq t + \ell$.

Finally, each party P_i gets the shares $\{\delta_{ij}, \hat{\delta}_{ij}\}_{j\in S}$. As evidenced by Lemma 4, the double-layer combination of Lagrange coefficients enables the perfect assembly of the correct result, if a sufficient number of shares are available.

Lemma 4. *For any subset S with $|S| \geq t+\ell$ and each $\nu \in [\ell]$, we have $\sum_{i,j\in S} \lambda_i^{(\nu)} \lambda_j^{(\nu)} \delta_{ij} = \gamma^{(\nu)} k^{(\nu)}$ and $\sum_{i,j\in S} \lambda_i^{(\nu)} \lambda_j^{(\nu)} \hat{\delta}_{ij} = \gamma^{(\nu)} x^{(\nu)}$. (The proof is deferred to Appendix B.2.)*

Online phase The batch size for the online phase is ℓ, where each execution consumes ℓ values, $\{R^{(\mu\nu)}\}_{\nu\in[\ell]}$ for some μ. A detailed description is shown in Figure 3.

To avoid a polynomial multiplication, the s-component is rewritten as $s = r \cdot \frac{\gamma(mr^{-1}+x)}{\gamma k} \bmod q$. Specifically, we use a polynomial $p(\cdot)$ to represent ℓ values mr^{-1}'s, which requires ℓ messages to be provided as input simultaneously. For security, each a_{ij} and b_{ij} is masked by a share of zero. Adopting the technique from [48], we utilize the fact in Lemma 5 to guarantee the correctness of P_i's $\{a_{ij}, b_{ij}\}_j$. Public verifiability is guaranteed by a $\mathcal{R}^m_{\mathsf{DDH}}$-relation proof π'_i that attests to whether $\{R^{(\nu)}, A_i^{(\nu)} = \gamma_i R^{(\nu)}, C^{(\nu)}, B_i^{(\nu)} = \gamma_i C^{(\nu)}\}_{\nu\in[\ell]}$ are ℓ DDH tuples. This ZKP is aggregatable, and we defer its concrete construction to Appendix A.1.

Lemma 5. *For any subset S with $|S| \geq t+\ell$ and each $\nu \in [\ell]$, we have $\sum_{j\in S} \lambda_j^{(\nu)} (a_{ij} + \beta_{ji} - \beta_{ij}) = \gamma_i k^{(\nu)}$ and $\sum_{j\in S} \lambda_j^{(\nu)} (b_{ij} + \hat{\beta}_{ji} - \hat{\beta}_{ij}) = \gamma_i (p(-\nu+1) + x)$. (The proof is deferred to Appendix B.3.)*

*Remark 3 (**Signing one by one**).* 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. The trade-off is that it prevents amortization during the online phase.

Fig. 3: Online Phase of Signing Protocol

For each $\mu \in [\hat{\ell}]$, let $k_i = k_i^{(\mu)}$, P_i lets $R^{(\nu)} = R^{(\mu\nu)}, r^{(\nu)} = \mathsf{F}(R^{(\nu)}), \forall \nu \in [\ell]$, and retrieves the selected γ_i, the generated $\{\delta_{ij}, \hat{\delta}_{ij}\}_{j \neq i}$, and the received $\{B_{ij}, \hat{B}_{ij}\}_{i,j \in S}$ during the μ-th call of VNIM. For each $\mu \in [\hat{\ell}]$, the parties can generate ℓ signatures for ℓ message $\{\mathsf{msg}^{(\nu)}\}_{\nu \in [\ell]}$. Let Q be the remaining (semi-)honest parties after the offline phase.

One-Round Sending.

- With $m^{(\nu)} = \mathsf{H}(\mathsf{msg}^{(\nu)})$, each party P_i interpolates the fixed degree-$(\ell-1)$ polynomial $p(\cdot)$ with $p(-\nu+1) = m^{(\nu)}(r^{(\nu)})^{-1}, \forall \nu \in [\ell]$.
- Pick two random degree-$(t+\ell-1)$ masking polynomials $z(\cdot), \hat{z}(\cdot)$ with $z(-\nu+1) = \hat{z}(-\nu+1) = 0, \forall \nu \in [\ell]$.
- $a_{ii} = \gamma_i k_i + z(i) \bmod q$, $\{a_{ij} = \delta_{ij} + z(j) \bmod q\}_{j \in S \setminus \{i\}}$, $b_{ii} = \gamma_i(p(i) + x_i) + \hat{z}(i) \bmod q$, $\{b_{ij} = \gamma_i p(j) + \hat{\delta}_{ij} + \hat{z}(j) \bmod q\}_{j \in S \setminus \{i\}}$.
- $A_i^{(\nu)} = \gamma_i R^{(\nu)}$, $C^{(\nu)} = p(-\nu+1)G + X$, $B_i^{(\nu)} = \gamma_i C^{(\nu)}$, generate an aggregated proof $\pi_i' \leftarrow \mathsf{Prove}_{\mathcal{R}_{\mathsf{DDH}}^m}(\{R^{(\nu)}, A_i^{(\nu)}, C^{(\nu)}, B_i^{(\nu)}\}_{\nu \in [\ell]}, \gamma_i)$.
- Send $(\{a_{ij}, b_{ij}\}_{j \in S}, \pi_i')$ to the aggregator.

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

1. Check the validity of P_i's shares $\{a_{ij}, b_{ij}\}$: for each $\nu \in [\ell]$, compute $A_i^{(\nu)} \coloneqq \sum_{j \in S} \lambda_j^{(\nu)}(a_{ij}G + B_{ji} - B_{ij})$, $B_i^{(\nu)} \coloneqq \sum_{j \in S} \lambda_j^{(\nu)}(b_{ij}G + \hat{B}_{ji} - \hat{B}_{ij})$, verify the proof $\mathsf{Verify}_{\mathcal{R}_{\mathsf{DDH}}^m}(\{R^{(\nu)}, A_i^{(\nu)}, C^{(\nu)}, B_i^{(\nu)}\}_{\nu \in [\ell]}, \pi_i') \to b$. If $b = 0$, set $S = S \setminus \{i\}$
2. If $|S| \geq t + \ell$, generate $s^{(\nu)} = r^{(\nu)} \cdot \frac{\sum_{i,j \in S} \lambda_i^{(\nu)} \lambda_j^{(\nu)} b_{ij}}{\sum_{i,j \in S} \lambda_i^{(\nu)} \lambda_j^{(\nu)} a_{ij}} \bmod q$. Output $\mathsf{sig}^{(\nu)} = (r^{(\nu)}, \mathsf{min}(s^{(\nu)}, q - s^{(\nu)}))$, $\forall\ \nu \in [\ell]$.

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 random polynomials $z(\cdot), \hat{z}(\cdot)$ are fresh for each message, and are now zero *only* at the point $-\nu + 1$. The generation of $\{a_{i,j}\}_j$ remains unchanged, but $\{b_{i,j}\}_j$ are changed: $b_{i,i} = \gamma_i(mr^{-1} + x_i) + \hat{z}(i), \{b_{i,j} = \gamma_i mr^{-1} + \hat{\delta}_{i,j} + \hat{z}(j)\}_j$.
2. Each party generates the proof π_i' to prove that $(R = rG, A_i = \gamma_i R, C = mr^{-1}G + X, B_i = \gamma_i C)$ is a DDH tuple, and the verifier can assemble A_i, B_i using $\{\lambda_j^{(\nu)}\}_j$.

For security, using the correct coefficients $\{\lambda_j^{(\nu)}\}_j$ can produce $m^{(\nu)}$'s signature, while using other Lagrange coefficients will yield a random value that conceals secret values, due to the Change 1.

4.3 Security Claim

Due to space limitations, we have deferred the specification of the ideal functionality $\mathcal{F}_{\mathsf{ECDSA}}$ for threshold ECDSA (Figure 4) and our security proof to the Appendix E.

Theorem 1. *If (1) VNIM satisfies correctness, privacy, and verifiability defined in Section 3.1; (2) PVSS and* PVSSeq *satisfy correctness, verifiability, and privacy defined in Section 2.3; (3)* $\mathcal{R}_{\mathsf{DL\text{-}D}}, \mathcal{R}_{\mathsf{DL\text{-}D}}^m, \mathcal{R}_{\mathsf{DDH}}^m$*-relation proofs are zero-knowledge proofs with standard soundness, our scheme realizes* $\mathcal{F}_{\mathsf{ECDSA}}$ *securely against a malicious adversary that statically corrupts up to t fixed parties.*

We provide a simulation-based security proof based on the security properties of our underlying primitives, including VNIM, PVSS, PVSSeq, and zero-knowledge proofs. Concretely, while the verifiability of VNIM and PVSS (as well as PVSSeq) generally requires zero-knowledge proofs of knowledge (ZKPoK) for $\mathcal{R}_{\mathsf{ecA}}$, $\mathcal{R}_{\mathsf{ecB}}$, and $\mathcal{R}_{\mathsf{key}}$, their requirements diverge in the threshold ECDSA protocol. Specifically, $\mathcal{R}_{\mathsf{key}}$ is used in key generation, while $\mathcal{R}_{\mathsf{ecA}}$ and $\mathcal{R}_{\mathsf{ecB}}$ are deployed during signing. Because the simulator extracts corrupted parties' secret shares k_i from the simulated PVSS process, the proof π_{Bk_i} for $\mathcal{R}_{\mathsf{ecB}}$ does not need to be a proof of knowledge, whereas $\pi_{A\gamma_i}$

Table 2: **The (amortized) communication costs of robust threshold ECDSAs.** We set $n = 2t + 1$ for WMC24 and TX25, and $n = 2t + \ell$ for our cases.

t	Offline communication (KB)						Online communication (KB)					
	2	4	8	12	16	32	2	4	8	12	16	32
WMC24 [49]	4.1	4.1	4.1	4.1	4.1	4.1	0.8	0.8	0.8	0.8	0.8	0.8
TX25 [48]	6.12	11.3	22	32	44	87	0.36	0.61	1.11	1.61	2.11	4.11
$\ell = 1$ - ARES	4.26	6.36	10.47	14.55	18.63	34.89	0.36	0.61	1.11	1.61	2.11	4.11
$\ell = \lfloor t/3 \rfloor$ - ARES	4.26	6.36	5.47	4	4.1	3.94	0.36	0.61	0.59	0.45	0.47	0.47
$\ell = \lfloor t/2 \rfloor$ - ARES	4.26	3.41	2.98	2.83	2.76	2.65	0.36	0.34	0.32	0.32	0.32	0.32

Table 3: **The (amortized) running times of robust threshold ECDSAs.** We set $n = 2t + 1$ for WMC24 and TX25, and $n = 2t + \ell$ for our cases.

t	Offline runtimes (ms)						Online runtimes (ms)					
	2	4	8	12	16	32	2	4	8	12	16	32
WMC24 [49]	528	990	1653	2395	3145	6137	238	404	784	1120	1456	2800
TX25 [48]	515	1172	2873	5089	7820	23902	1.7	3.3	10.8	18.4	35.6	122.9
$\ell = 1$ - ARES	863	1648	3638	6108	9234	28134	1.6	3.5	10.2	19.1	34.3	125.7
$\ell = \lfloor t/3 \rfloor$ - ARES	865	1647	1964	1755	2170	3504	1.7	3.5	10.1	20.8	38.1	124
$\ell = \lfloor t/2 \rfloor$ - ARES	867	910	1115	1321	1534	2488	1.7	4.1	12.8	28.2	39.2	118.6

for $\mathcal{R}_{\mathsf{ecA}}$ strictly does. To guarantee concurrent Universal Composability (UC) security without rewinding during signing, $\pi_{A\gamma_i}$ should be instantiated via the Fischlin transform, though our concrete implementation employs the lightweight Fiat-Shamir heuristic for practical performance.

5 Performance

We mainly compare our scheme with existing robust schemes, including WMYC23 [50], MWC24 [49], and TX25 [48]. Since TX25 utilizes similar techniques to WMYC23 but outperforms it, we focus our comparison on MWC24 and TX25.

Testbed Environment. Experiments were run on Windows 11 (WSL Ubuntu 24.04 LTS), Intel Core Ultra 9 275HX, 64 GB RAM. 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.

Theoretical Sending Communication Costs. Our work involves two batch parameters: one is the packing parameter ℓ for packed secret sharing, and the other $\hat{\ell}$ is the number of honest signers. Since our robust protocol requires $\hat{n} \geq 2t + \ell$ where $\hat{n}$ is the number of signers, we can always set $\hat{\ell} = t + \ell$.

The offline phase consists of two components: the batch generation of $(t + \ell)\ell$ values of R via invoking the PVSS scheme, and the batch computation of ℓ multiplications leveraging NIM. The total sending communication cost of generating these R's in batch is $0.89t + 0.46\ell - 0.09$ kilobytes (KB) for each party. The amortized cost is $\frac{0.89t+0.46\ell-0.09}{(t+\ell)\ell}$ KB. In VNIMs, the communication overhead per party is constant at $1.02t + 0.51\ell + 1.01$ KB, yielding an amortized cost of $\frac{1.02t+0.51\ell+1.01}{\ell}$ KB. In total, we get

$$\frac{0.89t + 0.46\ell - 0.09}{(t + \ell)\ell} + \frac{1.02t + 0.51\ell + 1.01}{\ell} \text{ KB}.$$

The online phase requires sending $0.13t + 0.063\ell + 0.047$ KB of messages for each party. When we amortize this cost over ℓ signatures, the resulting online communication cost per signature is $\frac{0.13t+0.063\ell+0.047}{\ell}$ KB.

Table 2 demonstrates the (amortized) communication costs of WMC24, TX25, and ours. Even at $\ell = 1$, ARES maintains a linear communication complexity, but it achieves approximately a 50% improvement over TX25 (which is also linear) due to the use of super-invertible matrices and verifiable non-interactive multiplication. When $\ell = \lfloor t/3 \rfloor$ and $t \geq 12$, the offline and online amortized communication of ARES both reach an approximately constant level, surpassing WMC24 due to the packed secret sharing. At $\ell = \lfloor t/2 \rfloor$, the advantage becomes even more pronounced, as the communication cost is amortized down to an even smaller constant.

Running Times. As evidenced by the concrete benchmarks in Table 3, ARES achieves an online-friendly design, with online computational efficiency comparable to TX25 and significantly outperforms WMC24. In fact, when setting $\ell = 1$, the online phase of ARES essentially coincides with that of TX25. In the offline phase, ARES incurs slightly higher computation than TX25 for $\ell = 1$, likely because evaluating the coset labeling function $\phi(\cdot)$ in VNIM is marginally more expensive than the other class-group operations used in TX25. However, VNIM brings notable improvements in communication efficiency, making it a compelling trade-off. When employing packed secret sharing and setting $\ell \in \{t/3, t/2\}$, the amortized computational overhead drops significantly below that of TX25. Moreover, once t ≥ 12, its amortized overhead also becomes lower than that of WMC24.

References

1. Damiano Abram, Ivan Damgård, Claudio Orlandi, and Peter Scholl. An algebraic framework for silent preprocessing with trustless setup and active security. In *CRYPTO*, pages 421–452. Springer, 2022.
2. Damiano Abram, Lawrence Roy, and Peter Scholl. Succinct homomorphic secret sharing. In *EUROCRYPT*, pages 301–330. Springer, 2024.
3. 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.
4. Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure MPC with linear communication complexity. In *TCC*, volume 4948, pages 213–230. Springer, 2008.
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. BitGo. Ethereum wallet management. `https://www.bitgo.com/resources/eth-wallet-management`, 2026. Accessed: 2026-04-30.
8. Dan Boneh, Iftach Haitner, Yehuda Lindell, and Gil Segev. Exponent-vrfs and their applications. In *EUROCRYPT*, pages 195–224. Springer, 2025.
9. Elette Boyle, Lalita Devadas, and Sacha Servan-Schreiber. Non-interactive distributed point functions. In *PKC*, volume 15674, pages 3–35. Springer, 2025.
10. Luís TAN Brandão and René Peralta. Nist first call for multi-party threshold schemes. `https://nvlpubs.nist.gov/nistpubs/ir/2025/NIST.IR.8214C.2pd.pdf`, 2023.
11. 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.
12. 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.
13. Ignacio Cascudo and Bernardo David. SCRAPE: scalable randomness attested by public entities. In *ACNS*, volume 10355, pages 537–556. Springer, 2017.
14. 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.
15. 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.
16. Guilhem Castagnos and Fabien Laguillaumie. Linearly homomorphic encryption from DDH. In *CT-RSA*, volume 9048, pages 487–505. Springer, 2015.

17. Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *STOC*, pages 364–369. ACM, 1986.
18. Ran Cohen, Jack Doerner, Yashvanth Kondi, and Abhi Shelat. Secure multiparty computation with identifiable abort from vindicating release. *IACR Cryptol. ePrint Arch.*, page 1136, 2023.
19. Ran Cohen and Yehuda Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. *Journal of Cryptology*, 30(4):1157–1186, 2017.
20. Hila Dahari-Garbian, Ariel Nof, and Luke Parker. Trout: Two-round threshold ecdsa from class groups. In *CCS*, pages 380–393, 2025.
21. William M Daley and Raymond G Kammer. Digital signature standard (DSS). *BOOZ-ALLEN AND HAMILTON INC MCLEAN VA*, 2000.
22. Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *CRYPTO*, volume 4622, pages 572–590. Springer, 2007.
23. 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.
24. 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.
25. Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Threshold ECDSA in three rounds. In *S&P*, pages 3053–3071. IEEE, 2024.
26. 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.
27. Daniel Escudero, Yashvanth Kondi, Yifan Song, and Hernán Vanegas. L for the price of one: On the benefits of using more than t+1 parties in threshold signing. *IACR Cryptol. ePrint Arch.*, 2026:61, 2026.
28. 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.
29. Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In *CRYPTO*, volume 3621, pages 152–168. Springer, 2005.
30. Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In *CCS*, pages 1179–1194. ACM, 2018.
31. Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal dsa/ecdsa signatures and an application to bitcoin wallet security. In *ACNS*, pages 156–174. Springer, 2016.
32. Martin Hirt and Jesper Buus Nielsen. Robust multiparty computation with linear communication complexity. In *CRYPTO*, volume 4117, pages 463–482. Springer, 2006.
33. Bowen Jiang, Guofeng Tang, and Haiyang Xue. Three-round (robust) threshold ECDSA from threshold CL encryption. In *ACISP*, pages 224–244. Springer, 2025.
34. 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*, pages 4286–4300, 2024.
35. Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In *TCC*, volume 7785, pages 477–498. Springer, 2013.
36. Jonathan Katz and Antoine Urban. Honest-majority threshold ECDSA with batch generation of key-independent presignatures. *IACR Commun. Cryptol.*, 2(1):8, 2025.
37. Yehuda Lindell. Fast secure two-party ECDSA signing. In *CRYPTO*, volume 10402, pages 613–644. Springer, 2017.
38. 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.
39. 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.
40. Yingjie Lyu, Zengpeng Li, Hong-Sheng Zhou, and Xudong Deng. Threshold ECDSA in two rounds. In *CCS*, pages 2937–2950. ACM, 2025.
41. Yingjie Lyu, Zengpeng Li, Hong-Sheng Zhou, Haiyang Xue, Mei Wang, Shuchao Wang, and Mengling Liu. Bandwidth-efficient robust threshold ECDSA in three rounds. *IACR Cryptol. ePrint Arch.*, 2025.
42. Ariel Nof and Luke Parker. Trout++: Robust asynchronous two-round ECDSA for arbitrary thresholds. *IACR Cryptol. ePrint Arch.*, 2026:1455, 2026.
43. Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch, and Dominique Schröder. ROAST: robust asynchronous schnorr threshold signatures. In *CCS*, pages 2551–2564. ACM, 2022.
44. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO*, volume 435, pages 239–252. Springer, 1989.
45. Victor Shoup. Practical threshold signatures. In *EUROCRYPT*, volume 1807, pages 207–220. Springer, 2000.
46. 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.

47. Guofeng Tang, Tian Qiu, Bowen Jiang, Haiyang Xue, Guomin Yang, Man Ho Au, Robert H. Deng, and Kwok-Yan Lam. Robot: Robust Threshold BBS+ in Two Rounds. IEEE, 2026.
48. 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.
49. 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.
50. 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.
51. 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.
52. Haiyang Xue, Man Ho Au, Xiang Xie, Tsz Hon Yuen, and Handong Cui. Efficient online-friendly two-party ECDSA signature. In *CCS*, pages 558–573. ACM, 2021.
53. 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.

A Zero-Knowledge Proofs

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.

For a zero-knowledge proof of knowledge, a proof-of-knowledge property is required, which is stronger 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 $(x, w') \in \mathcal{R}$.

A.1 NIZK Instantiations

Each construction below is a Σ-protocol compiled to an NIZK via the Fiat-Shamir transform [28], with $e \in \mathbb{Z}_q$ denoting the hash challenge. Let $B_r := 2^{2\lambda_s} \tilde{s}\, q$ and $B_v := 2^{\lambda_s} q^2$ be the range bounds, where λ_s is the statistical security parameter and $\tilde{s} := \lceil \sqrt{|\Delta_K|} \rceil$ is the upper bound of $\mathcal{G}^q$'s order, with Δ_K the class group discriminant.

NIZK for $\mathcal{R}_{\mathsf{key}}$. Please refer to [53, Algorithm 2] for this proof.

NIZK for $\mathcal{R}_{\mathsf{DL\text{-}D}}$. Statement $(\mathcal{C}_a = (c_1, c_2), ek, A)$, witness (a, dk) s.t. $c_2 = \mathfrak{f}^a c_1^{dk}$, $ek = \mathfrak{g}_q^{dk}$, and $A = aG$, where $c_1, ek \in \mathcal{G}^q$, $c_2 \in \mathcal{G}$, and $A \in \mathbb{G}$.

- $\mathsf{Prove}((c_1, c_2, ek, A), (a, dk))$: Sample $u_a \leftarrow\!\!\$\ \mathbb{Z}_q$ and $u_{dk} \leftarrow\!\!\$\ [0, B_r]$. Compute $U_A := u_a G$, $U_{\mathsf{dec}} := \mathfrak{f}^{u_a} c_1^{u_{dk}}$, $ek_0 := \mathfrak{g}_q^{u_{dk}}$, and $e := \mathsf{H}(c_1 \| c_2 \| ek \| A \| U_A \| U_{\mathsf{dec}} \| ek_0)$. Output $(U_A, U_{\mathsf{dec}}, ek_0, s_a, s_{dk})$ where $s_a := u_a + ea \in \mathbb{Z}_q$ and $s_{dk} := u_{dk} + e \cdot dk$.
- $\mathsf{Verify}((c_1, c_2, ek, A), (U_A, U_{\mathsf{dec}}, ek_0, s_a, s_{dk}))$: Recompute e as above. Accept iff $s_a \leq q$, $s_{dk} \leq B_r$, and the following hold: $s_a G = U_A + eA$, $\mathfrak{f}^{s_a} c_1^{s_{dk}} = U_{\mathsf{dec}} \cdot c_2^e$, and $\mathfrak{g}_q^{s_{dk}} = ek_0 \cdot ek^e$.

NIZK for $\mathcal{R}^m_{\mathsf{DL\text{-}D}}$. Let $\mathcal{C}_i = (c_{i,1}, c_{i,2})$ for $i \in [\ell]$. The relation is $\mathcal{R}^m_{\mathsf{DL\text{-}D}} = \{((\{\mathcal{C}_i, A_i\}_{i\in[\ell]}, ek), (\{a_i\}_{i\in[\ell]}, dk)) : A_i = a_i G,\ c_{i,2} = \mathfrak{f}^{a_i} c_{i,1}^{dk}\ \forall i \in [\ell],\ ek = \mathfrak{g}_q^{dk}\}$. Equivalently, $a_i = \mathsf{CL.Dec}(dk, \mathcal{C}_i)$ and $A_i = a_i G$ for every $i \in [\ell]$. Here $c_{i,1}, ek \in \mathcal{G}^q$, $c_{i,2} \in \mathcal{G}$, and $A_i \in \mathbb{G}$.

The proof first aggregates ℓ instances into a single relation. Compute $\{e_i\}_{i\in[\ell]} \leftarrow \mathsf{H}_{\mathsf{fs}}(\{\mathcal{C}_i, A_i\}_{i\in[\ell]}, ek)$ and set

$$c_1 := \prod_{i\in[\ell]} c_{i,1}^{e_i}, \quad c_2 := \prod_{i\in[\ell]} c_{i,2}^{e_i}, \quad A := \sum_{i\in[\ell]} e_i A_i \in \mathbb{G}.$$

Then, invoke $\mathcal{R}_{\mathsf{DL\text{-}D}}$ on $(\mathcal{C} = (c_1, c_2), ek, A)$ with the witness $(\sum_{i\in[\ell]} e_i a_i \bmod q, ek)$.

NIZK for $\mathcal{R}_{\mathsf{share}}$. Statement $(\{ek_j, \mathcal{P}_j, \mathcal{C}_j\}_{j\in S})$, witness $(\{a_j, a'_j\}_{j\in S}, \rho)$ s.t. for all $j \in S$, $\mathcal{C}_j = (\mathfrak{R}, \mathfrak{B}_j)$ with $\mathfrak{R} = \mathfrak{g}_q^{\rho}$, $\mathfrak{B}_j = ek_j^{\rho}\mathfrak{f}^{a_j}$, and $\mathcal{P}_j = a_j G + a'_j H$, where $\mathfrak{g}_q$ is the generator of $\mathcal{G}^q$ and $\rho \in \mathcal{D}_q$ is shared across all j.

The proof first aggregates the $|S|$ parallel instances into a single relation. Compute $\{e_j\}_{j\in S} \leftarrow \mathsf{H}_{\mathsf{fs}}(\{\mathcal{C}_j, \mathcal{P}_j\}_{j\in S})$ and set

$$ek := \prod_{j\in S} ek_j^{e_j}, \quad \mathfrak{B} := \prod_{j\in S} \mathfrak{B}_j^{e_j}, \quad \mathcal{P} := \sum_{j\in S} e_j \mathcal{P}_j.$$

With $a := \sum_{j\in S} e_j a_j \bmod q$ and $a' := \sum_{j\in S} e_j a'_j \bmod q$, the relation reduces to

$$\mathcal{R}^* = \{((\mathfrak{R}, \mathfrak{B}, \mathcal{P}, ek), (a, a' \in \mathbb{Z}_q, \rho \in \mathcal{D}_q)) : \\ \mathfrak{R} = \mathfrak{g}_q^{\rho},\ \mathfrak{B} = ek^{\rho}\mathfrak{f}^{a},\ \mathcal{P} = aG + a'H\}.$$

- $\mathsf{Prove}((\mathfrak{R}, \mathfrak{B}, \mathcal{P}, ek), (a, a', \rho))$: Sample $\tilde{\rho} \leftarrow\$ [0, B_r]$ and $\tilde{a}, \tilde{a}' \leftarrow\$ \mathbb{Z}_q$. Compute $\tilde{\mathfrak{R}} := \mathfrak{g}_q^{\tilde{\rho}}$, $\tilde{\mathfrak{B}} := ek^{\tilde{\rho}}\mathfrak{f}^{\tilde{a}}$, $\tilde{\mathcal{P}} := \tilde{a}G + \tilde{a}'H$, and $e := \mathsf{H}(\mathfrak{R}\|\mathfrak{B}\|\mathcal{P}\|ek\|\tilde{\mathfrak{R}}\|\tilde{\mathfrak{B}}\|\tilde{\mathcal{P}})$. Output $(\tilde{\mathfrak{R}}, \tilde{\mathfrak{B}}, \tilde{\mathcal{P}}, s_\rho, s_a, s_{a'})$ where $s_\rho := \tilde{\rho} + e\rho$, $s_a := \tilde{a} + ea \mod q$, and $s_{a'} := \tilde{a}' + ea' \mod q$.
- Verify: Recompute $\{e_j\}$ and $(ek, \mathfrak{B}, \mathcal{P})$ as above, then recompute e. Accept iff $s_\rho \le B_r$, $s_a, s_{a'} \le q$, and the following hold: $\mathfrak{g}_q^{s_\rho} = \tilde{\mathfrak{R}} \cdot \mathfrak{R}^e$, $ek^{s_\rho}\mathfrak{f}^{s_a} = \tilde{\mathfrak{B}} \cdot \mathfrak{B}^e$, and $s_a G + s_{a'} H = \tilde{\mathcal{P}} + e\mathcal{P}$.

NIZK for $\mathcal{R}_{\mathsf{ecA}}$. Recall that $\mathcal{R}_{\mathsf{ecA}} = \{((\mathsf{ec}_A, X_A), (x_A, k_A, r_A)) : X_A = x_A G + k_A H,\ \mathsf{ec}_A = \mathfrak{g}_0^{r_A}\mathfrak{g}_1^{x_A}\}$. The public statement is (ec_A, X_A) and the witness is (x_A, k_A, r_A).

- $\mathsf{Prove}((\mathsf{ec}_A, X_A), (x_A, k_A, r_A))$: Sample $\tilde{x}_A \leftarrow\$ [0, B_v], \tilde{k}_A \leftarrow\$ \mathbb{Z}_q, \tilde{r}_A \leftarrow\$ [0, B_r]$. Compute $\widetilde{\mathsf{ec}}_A := \mathfrak{g}_0^{\tilde{r}_A}\mathfrak{g}_1^{\tilde{x}_A}$ and $\widetilde{X}_A := \tilde{x}_A G + \tilde{k}_A H$. Compute $e := \mathsf{H}(\mathsf{ec}_A\|X_A\|\widetilde{\mathsf{ec}}_A\|\widetilde{X}_A)$. Output $\pi_A := (\widetilde{\mathsf{ec}}_A, \widetilde{X}_A, s_{x_A}, s_{k_A}, s_{r_A})$, where $s_{x_A} := \tilde{x}_A + ex_A$, $s_{k_A} := \tilde{k}_A + ek_A \bmod q$, and $s_{r_A} := \tilde{r}_A + er_A$.
- $\mathsf{Verify}((\mathsf{ec}_A, X_A), \pi_A)$: Parse $\pi_A = (\widetilde{\mathsf{ec}}_A, \widetilde{X}_A, s_{x_A}, s_{k_A}, s_{r_A})$. Recompute e as above. Accept iff $s_{x_A} \le B_v, s_{r_A} \le B_r$, $\mathfrak{g}_0^{s_{r_A}}\mathfrak{g}_1^{s_{x_A}} = \widetilde{\mathsf{ec}}_A \cdot \mathsf{ec}_A^e$ and $s_{x_A} G + s_{k_A} H = \widetilde{X}_A + eX_A$.

NIZK for $\mathcal{R}_{\mathsf{ecB}}$. Recall that $\mathcal{R}_{\mathsf{ecB}} = \{((\mathsf{ec}_B, X_B), (x_B, r_B)) : X_B = x_B G,\ \mathsf{ec}_B = (\mathfrak{g}_0^{r_B}, \mathfrak{f}^{x_B}\mathfrak{g}_1^{r_B})\}$. The public statement is (ec_B, X_B) and the witness is (x_B, r_B). Write $\mathsf{ec}_B = (c_{B,0}, c_{B,1})$.

- $\mathsf{Prove}((\mathsf{ec}_B, X_B), (x_B, r_B))$: Sample $\tilde{x}_B \leftarrow\$ \mathbb{Z}_q, \tilde{r}_B \leftarrow\$ [0, B_r]$. Compute $\widetilde{\mathsf{ec}}_B := (\tilde{c}_{B,0}, \tilde{c}_{B,1}) := (\mathfrak{g}_0^{\tilde{r}_B}, \mathfrak{f}^{\tilde{x}_B}\mathfrak{g}_1^{\tilde{r}_B})$ and $\widetilde{X}_B := \tilde{x}_B G$. Compute $e := \mathsf{H}(\mathsf{ec}_B\|X_B\|\widetilde{\mathsf{ec}}_B\|\widetilde{X}_B)$. Output $\pi_B := (\widetilde{\mathsf{ec}}_B, \widetilde{X}_B, s_{x_B}, s_{r_B})$, where $s_{x_B} := \tilde{x}_B + ex_B \bmod q$ and $s_{r_B} := \tilde{r}_B + er_B$.
- $\mathsf{Verify}((\mathsf{ec}_B, X_B), \pi_B)$: Parse $\pi_B = (\widetilde{\mathsf{ec}}_B, \widetilde{X}_B, s_{x_B}, s_{r_B})$, $\widetilde{\mathsf{ec}}_B = (\tilde{c}_{B,0}, \tilde{c}_{B,1})$, and $\mathsf{ec}_B = (c_{B,0}, c_{B,1})$. Recompute e as above. Accept iff $s_{r_B} \le B_r$, $\mathfrak{g}_0^{s_{r_B}} = \tilde{c}_{B,0} \cdot c_{B,0}^e$, $\mathfrak{f}^{s_{x_B}}\mathfrak{g}_1^{s_{r_B}} = \tilde{c}_{B,1} \cdot c_{B,1}^e$, and $s_{x_B} G = \widetilde{X}_B + eX_B$.

NIZK for $\mathcal{R}_{\mathsf{outA}}$. Recall that $\mathcal{R}_{\mathsf{outA}} = \{((S_A, \mathfrak{h}^*, \mathsf{ec}_A, \mathsf{ec}_B = (c_0, c_1)), (s_A, x_A, r_A)) : \mathfrak{h}^* = c_0^{r_A} c_1^{x_A}\mathfrak{f}^{-s_A},\ \mathsf{ec}_A = \mathfrak{g}_0^{r_A}\mathfrak{g}_1^{x_A},\ S_A = s_A G\}$. The public statement is $(S_A, \mathfrak{h}^*, \mathsf{ec}_A, \mathsf{ec}_B = (c_0, c_1))$ and the witness is (s_A, x_A, r_A).

- $\mathsf{Prove}((S_A, \mathfrak{h}^*, \mathsf{ec}_A, \mathsf{ec}_B), (s_A, x_A, r_A))$: Parse $\mathsf{ec}_B = (c_0, c_1)$. Sample $\tilde{s}_A \leftarrow\$ \mathbb{Z}_q, \tilde{x}_A \leftarrow\$ [0, B_v], \tilde{r}_A \leftarrow\$ [0, B_r]$. Compute $\widetilde{\mathfrak{h}}_A := c_0^{\tilde{r}_A} c_1^{\tilde{x}_A}\mathfrak{f}^{-\tilde{s}_A}$, $\widetilde{\mathsf{ec}}_A := \mathfrak{g}_0^{\tilde{r}_A}\mathfrak{g}_1^{\tilde{x}_A}$, and $\widetilde{S}_A := \tilde{s}_A G$. Compute $e := \mathsf{H}(S_A\|\mathfrak{h}^*\|\mathsf{ec}_A\|\mathsf{ec}_B\|\widetilde{\mathfrak{h}}_A\|\widetilde{\mathsf{ec}}_A\|\widetilde{S}_A)$. Output $\pi_{\mathsf{outA}} := (\widetilde{\mathfrak{h}}_A, \widetilde{\mathsf{ec}}_A, \widetilde{S}_A, z_{s_A}, z_{x_A}, z_{r_A})$, where $z_{s_A} := \tilde{s}_A + es_A \bmod q$, $z_{x_A} := \tilde{x}_A + ex_A$, and $z_{r_A} := \tilde{r}_A + er_A$.
- $\mathsf{Verify}((S_A, \mathfrak{h}^*, \mathsf{ec}_A, \mathsf{ec}_B), \pi_{\mathsf{outA}})$: Parse $\mathsf{ec}_B = (c_0, c_1)$ and $\pi_{\mathsf{outA}} = (\widetilde{\mathfrak{h}}_A, \widetilde{\mathsf{ec}}_A, \widetilde{S}_A, z_{s_A}, z_{x_A}, z_{r_A})$. Recompute e as above. Accept iff $z_{x_A} \le B_v, z_{r_A} \le B_r$, $c_0^{z_{r_A}} c_1^{z_{x_A}}\mathfrak{f}^{-z_{s_A}} = \widetilde{\mathfrak{h}}_A \cdot (\mathfrak{h}^*)^e$, $\mathfrak{g}_0^{z_{r_A}}\mathfrak{g}_1^{z_{x_A}} = \widetilde{\mathsf{ec}}_A \cdot \mathsf{ec}_A^e$, and $z_{s_A} G = \widetilde{S}_A + eS_A$.

NIZK for $\mathcal{R}_{\mathsf{outB}}$. Recall that $\mathcal{R}_{\mathsf{outB}} = \{((S_B, \mathfrak{h}^*, \mathsf{ec}_A, c_0), (s_B, r_B)) : \mathfrak{h}^* = \mathsf{ec}_A^{r_B}\mathfrak{f}^{s_B},\ c_0 = \mathfrak{g}_0^{r_B},\ S_B = s_B G\}$. The public statement is $(S_B, \mathfrak{h}^*, \mathsf{ec}_A, c_0)$ and the witness is (s_B, r_B).

- $\mathsf{Prove}((S_B, \mathfrak{h}^*, \mathsf{ec}_A, c_0), (s_B, r_B))$: Sample $\tilde{s}_B \leftarrow\$ \mathbb{Z}_q, \tilde{r}_B \leftarrow\$ [0, B_r]$. Compute $\widetilde{\mathfrak{h}}_B := \mathsf{ec}_A^{\tilde{r}_B} \mathfrak{f}^{\tilde{s}_B}$, $\tilde{c}_0 := \mathfrak{g}_0^{\tilde{r}_B}$, and $\widetilde{S}_B := \tilde{s}_B G$. Compute $e := \mathsf{H}(S_B \| \mathfrak{h}^* \| \mathsf{ec}_A \| c_0 \| \widetilde{\mathfrak{h}}_B \| \tilde{c}_0 \| \widetilde{S}_B)$. Output $\pi_{\mathsf{outB}} := (\widetilde{\mathfrak{h}}_B, \tilde{c}_0, \widetilde{S}_B, z_{s_B}, z_{r_B})$, where $z_{s_B} := \tilde{s}_B + e s_B$ and $z_{r_B} := \tilde{r}_B + e r_B$.
- $\mathsf{Verify}((S_B, \mathfrak{h}^*, \mathsf{ec}_A, c_0), \pi_{\mathsf{outB}})$: Parse $\pi_{\mathsf{outB}} = (\widetilde{\mathfrak{h}}_B, \tilde{c}_0, \widetilde{S}_B, z_{s_B}, z_{r_B})$. Recompute e as above. Accept iff $z_{r_B} \le B_r$, $\mathsf{ec}_A^{z_{r_B}} \mathfrak{f}^{z_{s_B}} = \widetilde{\mathfrak{h}}_B \cdot (\mathfrak{h}^*)^e$, $\mathfrak{g}_0^{z_{r_B}} = \tilde{c}_0 \cdot c_0^e$, and $z_{s_B} G = \widetilde{S}_B + e S_B$.

NIZK for $\mathcal{R}^m_{\mathsf{DDH}}$. The statement is ℓ DDH tuples $\{R^{(\nu)}, A_i^{(\nu)} = \gamma_i R^{(\nu)}, C^{(\nu)}, B_i^{(\nu)} = \gamma_i C^{(\nu)}\}_{\nu \in [\ell]}$. The proof first aggregates the ℓ tuples into a single tuple. Compute $e \leftarrow \mathsf{H}_{\mathsf{fs}}(\{R^{(\nu)}, A_i^{(\nu)}, C^{(\nu)}, B_i^{(\nu)}\}_{\nu \in [\ell]})$ and set

$$A_i := \sum_{\nu \in [\ell]} e^\nu A_i^{(\nu)}, \quad B_i := \sum_{\nu \in [\ell]} e^\nu B_i^{(\nu)}, \quad \gamma = \sum_{\nu \in [\ell]} e^\nu \gamma_i \bmod q.$$

Invoke the well-studied Schnorr proof [44] to prove that $(R^{(\nu)}, A_i, C^{(\nu)}, B_i)$ is a DDH tuple.

B Deferred Proofs

B.1 Proof of Lemma 1

Proof. We have to show that for any index set $I \subseteq [\hat{n}]$ with $|I| = \ell$, the sub-matrix $\hat{\boldsymbol{\Lambda}}_S \in \mathbb{Z}_q^{\ell \times \ell}$ consisting of the columns of $i \in I$ of $\boldsymbol{\Lambda}_S$ is invertible. It is sufficient to show that the mapping defined by $\hat{\boldsymbol{\Lambda}}_S$ is surjective, i.e., for every $\boldsymbol{y} \in \mathbb{Z}_q^\ell$ there exists an $\boldsymbol{x}_I \in \mathbb{Z}_q^\ell$ such that $\boldsymbol{y} = \hat{\boldsymbol{\Lambda}}_S \boldsymbol{x}_I$. Equivalently, we show that for each $\boldsymbol{y} = (y_\nu)_{\nu \in [\ell]}$ there exists an $\boldsymbol{x} \in \mathbb{Z}_q^{\hat{n}}$ such that $\boldsymbol{y} = \boldsymbol{\Lambda}_S \boldsymbol{x}$ and $\boldsymbol{x}_{\bar{I}} = \boldsymbol{0}$, where $\boldsymbol{x}_{\bar{I}}$ denotes the sub-vector consisting of $\boldsymbol{x}$'s j-th elements with $j \in \bar{I} = [\hat{n}] \setminus I$. Note that the Lagrange coefficients in $\boldsymbol{\Lambda}_S$ are defined over the indices $\{-l+1, \cdots, 0\}$ and S. Without loss of generality, we assume that $S = [\hat{n}]$. Given the $\hat{n}$ points $\{(j, 0)\}_{j \in \bar{I}}$ and $\{(-\nu + 1, y_\nu)\}_{\nu \in [\ell]}$, a polynomial $f(\cdot)$ can be determined by Lagrange interpolation, and $\boldsymbol{x}_I$ can be generated as $f(i), i \in I$. Hence, $\hat{\boldsymbol{\Lambda}}_S$ is invertible. □

B.2 Proof of Lemma 4

Proof. Taking $\gamma^{(\nu)} k^{(\nu)}$ as an example, we decompose it into the following form.

$$\begin{aligned} \gamma^{(\nu)} k^{(\nu)} &= \left(\sum_{i \in S} \lambda_i^{(\nu)} \gamma_i \right) \left(\sum_{j \in S} \lambda_j^{(\nu)} k_j \right) \\ &\overset{\gamma_i k_j = \alpha_{ij} + \beta_{ji}}{=} \sum_{i,j \in S} \lambda_i^{(\nu)} \lambda_j^{(\nu)} (\alpha_{ij} + \beta_{ji}) \\ &= \sum_{i,j \in S} \lambda_i^{(\nu)} \lambda_j^{(\nu)} (\alpha_{ij} + \beta_{ij}) \\ &= \sum_{i,j \in S} \lambda_i^{(\nu)} \lambda_j^{(\nu)} \delta_{ij} \end{aligned} \tag{2}$$

The third equality holds due to $\sum_{i,j \in S} \lambda_i^{(\nu)} \lambda_j^{(\nu)} \beta_{ji} = \sum_{i,j \in S} \lambda_i^{(\nu)} \lambda_j^{(\nu)} \beta_{ij}$. □

B.3 Proof of Lemma 5

Proof.

$$\begin{aligned} \sum_{j \in S} \lambda_j^{(\nu)} (a_{ij} + \beta_{ji} - \beta_{ij}) &= \sum_{j \in S} \lambda_j^{(\nu)} (\alpha_{ij} + \beta_{ij} + \beta_{ji} - \beta_{ij} + z(i)) \\ &= \sum_{j \in S} \lambda_j^{(\nu)} (\alpha_{ij} + \beta_{ji}) \\ &= \sum_{j \in S} \lambda_j^{(\nu)} \gamma_i k_j = \gamma_i k^{(\nu)} \end{aligned}$$

$$\begin{aligned}\sum_{j\in S}\lambda_j^{(\nu)}(b_{ij}+\hat{\beta}_{ji}-\hat{\beta}_{ij}) &= \sum_{j\in S}\lambda_j^{(\nu)}(\gamma_i p(j)+\hat{\alpha}_{ij}+\hat{\beta}_{ji})\\ &= \sum_{j\in S}\lambda_j^{(\nu)}(\gamma_i p(j)+\gamma_i x_j)\\ &= \gamma_i(p(-\nu+1)+x)\end{aligned}$$

□

B.4 Security Proof for VNIM

Proof. Correctness follows naturally. Since $\mathfrak{h}^* = \phi(\mathfrak{b}) = \phi(\mathfrak{a})$ and $\mathfrak{a} = \mathfrak{b}\mathfrak{f}^{x_A x_B}$, we have $\frac{\mathfrak{a}}{\mathfrak{h}^*} \cdot \frac{\mathfrak{h}^*}{\mathfrak{b}} = \mathfrak{f}^{x_A x_B}$, which implies that $\mathsf{Dlog}(\mathfrak{a}/\mathfrak{h}^*) + \mathsf{Dlog}(\mathfrak{h}^*/\mathfrak{b}) = x_A x_B$.

We analyze the verifiability as follows. For $\rho \in \{A, B\}$, since π_ρ is a proof of knowledge x_ρ, we can construct an extractor $\mathcal{E}_1$ that extracts x_ρ from π_ρ s.t. $(x_\rho, X_\rho) \in \mathcal{R}_1$. Then, the correct s_ρ can be generated, as $s_\rho = x_{\bar{\rho}} x_\rho - s_{\bar{\rho}}$, where $x_{\bar{\rho}}, s_{\bar{\rho}}$ are the challenger's correct input and output for this execution. Next, we show that the adversary $\mathcal{A}$ cannot produce an incorrect S_ρ satisfying $s_\rho G \neq S_\rho$ alongside a valid proof τ_ρ. Given $\tau_\rho = (\mathfrak{h}^*, \tau'_\rho)$, verifying $\phi(\mathfrak{h}^*) = \mathfrak{h}^*$ ensures that $\mathfrak{h}^*$ is a valid representative element. The soundness of the proof τ'_ρ guarantees that the coset containing $\mathfrak{h}^*$ is correctly computed, and that S_ρ corresponds to the secret share underlying $\mathfrak{h}^*$, which must be s_ρ. Therefore, if the adversary outputs an incorrect S_ρ while the proof τ'_ρ passes verification, this directly contradicts its standard soundness.

Specifically, for privacy, we need to construct a simulator $\mathsf{Sim_A}$ to simulate $\mathsf{ec}_A, \pi_A, S_A, \tau_A$ and $\mathsf{Sim_B}$ to simulate $\mathsf{ec}_B, \pi_B, S_B, \tau_B$. $\mathsf{Sim_B}$ can be constructed as follows.

- $\mathsf{SimInput}_B(\mathsf{crs}, X_B)$: pick a random $x \leftarrow\!\$\, \mathbb{Z}_q$ compute $\mathsf{ec}_B := (\mathfrak{g}_0^r, \mathfrak{g}_1^r \mathfrak{f}^x)$ with a random r and then simulate π_B. Store $\mathsf{st}_B^* := (x, r)$.
- $\mathsf{SimOutput}_B(\mathsf{ec}_A, \pi_A, (x, r), X_B)$: If π_A is not valid, output $\perp$. Otherwise, extract x_A from π_A due to the knowledge soundness, run $\mathsf{Decode}_B(\mathsf{crs}, \mathsf{ec}_A, r, x) \to s'_B$, where $\mathfrak{h}^* = \phi(\mathfrak{c}^r)$ is generated. Then, generate the adversary's share $s_A = x \cdot x_A - s'_B \bmod q$ and simulate $S_B = x_A \cdot X_B - s_A G$. Finally, simulate the proof τ'_B and output $(S_B, \tau_B := (\mathfrak{h}^*, \tau'_B))$.

Due to the semantic security of CL encryption and the zero-knowledge property of proof π_B, the simulated (ec_B, π_B) is indistinguishable from the real one. At the same time, the simulated $\mathfrak{h}^*$ is identical to the representative element obtained by the adversary (as party A) in the real protocol. Since r is unknown to the adversary, given the public $\mathfrak{h}^*$, the adversary knows that $\mathfrak{c}^r$ is a preimage of $\mathfrak{h}^*$ under the function $\phi(\cdot)$. However, because there are q preimages in total, the adversary cannot determine which specific one it is. Therefore, the result s'_B produced by the simulation is perfectly hidden. The simulator can infer the adversary's additive share s_A based on s'_B, thereby simulating $S_B = x_A \cdot X_B - s_A G$. As a result, the simulated S_B follows the exact same distribution as the S_B generated by an honest party B holding the input x_B in the real protocol. Although the simulated S_B and $\mathfrak{h}^*$ will fail to satisfy the $\mathcal{R}_{\mathsf{outB}}$ relation, the adversary cannot detect this discrepancy due to the zero-knowledge property of the proof τ'_B.

The construction of the simulator Sim_A is very similar to that of Sim_B. One notable difference is that because X_A is a randomized commitment to x_A, it no longer maintains a verifiable linear relationship with S_A of the form $x_B X_A - s_B G = S_A$. Consequently, Sim_A can simulate S_A by simply sampling it at random. The indistinguishability analysis follows an analogous argument, relying on the hiding property of the Pedersen commitment over the class group (for ec_A) and the zero-knowledge property of the proofs π_A, τ'_A. □

C Security Properties and Construction of PVSS in [14]

In this section, we first formally define the security properties of PVSS, namely correctness, verifiability, and privacy.

Correctness. The honestly generated $(\{\mathcal{C}_i\}_{i\in[n]}, \pi_{\mathsf{Sh}})$ via $\mathsf{PVSS.Share}$ can pass the verification (i.e., $\mathsf{PVSS.VerifyShairng}(\cdot) \to 1$) with overwhelming probability.

Verifiability. With $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, a PVSS satisfies verifiability if for all PPT $\mathcal{A}$, $\Pr[\pi_{\mathsf{key}_i} \text{ is valid} \wedge \nexists\, dk_i \text{ s.t. } (ek_i, dk_i) \in \mathcal{R}_{\mathrm{key}} | (i, \mathsf{ek}_i, \pi_{\mathsf{key}_i}) \leftarrow \mathcal{A}(\mathsf{pp})] \leq \mathsf{negl}(\lambda)$ and $\Pr[\pi_{\mathsf{Sh}} \text{ is valid} \wedge \nexists \mathbf{s} \text{ s.t. } (\{\mathcal{C}_i\}_{i\in[n]}, \cdot) \leftarrow \mathsf{PVSS.Share}(\cdot, \cdot, \mathbf{s}) | ((\mathcal{C}_i)_{i\in[n]}, \pi_{\mathsf{Sh}}) \leftarrow \mathcal{A}(\mathsf{pp})] \leq \mathsf{negl}(\lambda)$.

Privacy. With $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, a PVSS satisfies verifiability if for all PPT $\mathcal{A}$ corrupting $t-1$ parties (w.l.o.g. $\mathcal{A}$ corrupts $P_i, i \in [t-1]$), we have the advantage of $\mathcal{A}$ winning in the game $\mathsf{Exp}_{\mathcal{A},\mathsf{PVSS}}$ is negligible with $\mathsf{Adv}_{\mathsf{Exp}_{\mathcal{A},\mathsf{PVSS}}} \leq \mathsf{negl}(\lambda)$. The game $\mathsf{Exp}_{\mathcal{A},\mathsf{PVSS}}$ is defined as follows:

1. For $i \in [t, n]$, the challenger runs $(dk_i, ek_i, \pi_{\mathsf{key}_i}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, i)$ and sends all created $(ek_i, \pi_{\mathsf{key}_i})$ to $\mathcal{A}$. $\mathcal{A}$ creates $\{ek_j, \pi_{\mathsf{key}_j}\}_{j\in[t-1]} \leftarrow \mathcal{A}(\mathsf{pp}, \{ek_i, \pi_{\mathsf{key}_i}\}_{i\in[t,n]})$ for the corrupted parties and sends them to the challenger, together with two values $\mathbf{s}_0, \mathbf{s}_1$.
2. If any of $\{\pi_{\mathsf{key}_j}\}_{j\in[t-1]}$ is rejected, the challenger sends $\perp$ to $\mathcal{A}$. Otherwise, the challenger picks $b \leftarrow\!\$\ \{0,1\}$, runs $(\{\mathcal{C}_i\}_{i\in[n]}, \pi_{\mathsf{Sh}}) \leftarrow \mathsf{PVSS.Share}(\mathsf{pp}, \{ek_i\}_{i\in[n]}, \mathbf{s}_b)$ (a sharing of $\mathbf{s}_b$), and sends $(\{\mathcal{C}_i\}_{i\in[n]}, \pi_{\mathsf{Sh}})$ to $\mathcal{A}$.
3. $\mathcal{A}$ wins if it outputs a guess $b' = b$.

In [14], the authors give a PVSS scheme over class groups based on the CL encryption. Let λ be the security parameter and $q > 2^\lambda$ prime. Let ℓ (size of secret), t (privacy threshold) and n (number of parties) be natural numbers, with $\ell, t, n = \mathrm{poly}(\lambda)$, $t \leq n \leq q$. We show the Setup, KeyGen, Share, $\mathsf{VerifySharing}$ algorithms below:

- $\mathsf{PVSS.Setup}(1^\lambda, q, \ell, t, n)$: let $\mathsf{pp}_{\mathsf{Sh}} := (q, \ell, t, n, G, H)$ where G, H are two independent generators of EC group $\mathbb{G}$, run $\mathsf{pp}_{\mathsf{CL}} := (q, \tilde{s}, \mathfrak{f}, \mathfrak{g}_q, \mathcal{F}, \mathcal{G}) \leftarrow \mathsf{CLGen}(1^\lambda, q)$, and output $\mathsf{pp} := (\mathsf{pp}_{\mathsf{Sh}}, \mathsf{pp}_{\mathsf{CL}})$
- $\mathsf{PVSS.KeyGen}(\mathsf{pp}, i)$, where $i \in [n]$:
  1. Generate $(ek_i, dk_i) \leftarrow \mathsf{CL.KGen}(\mathsf{pp}_{\mathsf{CL}})$, where $dk_i \leftarrow\!\$\ \mathcal{D}_q$ and $ek_i = \mathfrak{g}_q^{dk_i}$
  2. Generate the proof $\pi_{\mathsf{key}_i} \leftarrow \mathsf{Prove}_{\mathsf{key}}(ek_i, dk_i)$
  3. Output $(ek_i, dk_i, \pi_{\mathsf{key}_i})$.
- $\mathsf{PVSS.Share}(\mathsf{pp}, \{ek_j\}_{j\in[n]}, \mathbf{s})$, where $\mathbf{s} \in \mathbb{Z}_q^\ell$:
  1. Create a Shamir sharing of $\mathbf{s}$: sample a degree-$(t+\ell-1)$ polynomial $f(\cdot) \in \mathbb{Z}_q[X]$ s.t. $f(-\nu+1) = s_\nu$ for $\nu \in [\ell]$ and set $\sigma_j = f(j)$ for each $j \in [n]$.
  2. Sample $r \leftarrow\!\$\ \mathcal{D}_q$ and compute $\mathcal{R} = \mathfrak{g}_q^r$. For each ek_j, compute $\mathcal{B}_j = ek_j^r \cdot \mathfrak{f}^{\sigma_j}$, and set $\mathcal{C}_j = (\mathcal{R}, \mathcal{B}_j)$.
  3. Generate the proof π_{Sh} ensuring $\{\mathcal{C}_j\}_{j\in[n]}$ are valid encryptions of shares.
  4. Output $(\{\mathcal{C}_j\}_{j\in[n]}, \pi_{\mathsf{sh}})$.
- $\mathsf{PVSS.VerifySharing}(\mathsf{pp}, \{\mathcal{C}_j\}_{j\in[n]}, \pi_{\mathsf{Sh}})$, where $\mathcal{C}_j = (\mathcal{R}, \mathcal{B}_j)$: Run $\mathsf{Verify}_{\mathsf{Sh}}$ on π_{sh} with respect to statement $(\mathcal{R}, \{\mathcal{B}_j\}_{j\in[n]})$ and output its result.

For concrete construction of π_{Sh}, we refer the readers to [14, Figure 4].

D Dual-code-based Verification

To construct a publicly verifiable secret sharing, one can use dual-code-based verification [13]. Essentially, it uses a dual vector whose inner product with a vector of correct shares would be zero, but would be non-zero with a vector of inconsistent shares.

Lemma 6 ([13], Lemma 1). *Define C and its dual $C^\perp$ as $C := \{(f(1), f(2), \cdots, f(n)) : f(z) \in \mathbb{Z}_q[z], \mathsf{deg}(f(z)) \leq t-1\}$ and $C^\perp := \{(v_1 p(1), v_2 p(2), \cdots, v_n p(n)) : p(z) \in \mathbb{Z}_q[z], \mathsf{deg}(p(z)) \leq n-t-1\}$, with $v_i := \prod_{j\in[n]\setminus\{i\}} \frac{1}{i-j}$. For a vector of inconsistent shares $(x_1, \cdots, x_n) \in \mathbb{Z}_q^n \setminus C$, and any dual vector $(c_1^\perp, \cdots, c_n^\perp)$ chosen uniformly from $C^\perp$, the probability that $\sum_{i=1}^n x_i c_i^\perp = 0$ is $1/q$.*

Lemma 6 presents a verification method: if the set of shares $(x_1, x_2, \cdots, x_n)$ is generated from a polynomial with degree $\geq t$, then this set of shares passes the verification $\sum_{i=1}^n x_i c_i^\perp = 0$ with negligible probability.

We define an algorithm $\mathsf{SS.Dual}$ that returns dual vector $\{x_i^\perp\}_{i\in[n]}$ in $C^\perp$, where $[n] = \{1, \cdots, n\}$ is the set of evaluation points of the polynomial, and can be replaced with any set with size $\geq t$.

- $\mathsf{SS.Dual}([n], t) \to \{x_i^\perp\}_{i\in[n]}$: Choose a random polynomial $p(z) \leftarrow\!\$\ \mathbb{Z}_q[z]$ with degree $n-t-1$, return $x_i^\perp := v_i p(i)$ for $i \in [n]$, where $v_i = \prod_{j\in[n]\setminus\{i\}} \frac{1}{i-j}$.

Public verification in exponent: The dual code can be used to publicly verify the Pedersen commitments $\{\mathcal{P}_{f(i)} = f(i)G + f'(i)H\}_{i\in[n]}$ of share $f(i)$, where $(G, H) \in \mathbb{G}^2$ is Pedersen's public parameter.

Lemma 7 ([49], Lemma 3). *Let $\mathcal{P}_{f(i)} = f(i)G+f'(i)H$ be a Pedersen commitment of share $f(i)$ using randomness $f'(i)$. For $\{x_i^{\perp}\}_{i\in[n]} \leftarrow \mathsf{SS.Dual}([n], t)$, if $\deg(f(z)) > t-1$ or $\deg(f'(z)) > t-1$, then the probability that $\sum_{i\in[n]} x_i^{\perp} \cdot \mathcal{P}_{f(i)} = O$ is $1/q$, where $O \in \mathbb{G}$ is the identity element of $\mathbb{G}$.*

E Security Proof of Theorem 1

We first illustrate the threshold ECDSA functionality $\mathcal{F}_{\mathsf{ECDSA}}$ in Figure 4. It divides the signing process into an offline phase that generates $\ell\hat{\ell}$ R's in batch and an online phase that generates ℓ signatures in batch.

Fig. 4: 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. The ideal adversary Sim can corrupt up to t parties that are indexed by $\mathcal{P}^*$. This functionality ensures synchrony within the framework of Katz et al. [35], meaning it advances from one round to the next only 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), send X as the public key to Sim
  2. if receiving $(\mathsf{exposed}, sid, j)$ from Sim, remove party P_j from the party set, send X as the public key to all parties, and ignore any further call to keygen.

Offline Signing.

- On receiving $(\mathsf{offsign}, sid)$ from some party P_i, if keygen was already called and sid is fresh, send $(\mathsf{offsign}, sid, i)$ to Sim.
- After receiving $(\mathsf{offsign}, sid)$ from $\geq t + \ell$ parties, let set S denote the singer set and let $\hat{\ell} = |S| - t$, sample $k^{(\mu\nu)} \leftarrow\!\!\$\, \mathbb{Z}_q$ and compute $R^{(\mu\nu)} = k^{(\mu\nu)}G$ for each $\mu \in [\hat{\ell}], \nu \in [\ell]$.
- Send $\{R^{(\mu\nu)}\}_{\mu\in[\hat{\ell}],\nu\in[\ell]}$ to Sim. If receiving $(\mathsf{exposed}, sid, j)$ from Sim, remove party P_j from the signer set S.
- If $|S| \geq t + \ell$, send $\{R^{(\mu\nu)}\}_{\mu\in[\hat{\ell}],\nu\in[\ell]}$ to all signers in S and store $(\mathsf{offsigned}, sid)$ in memory.

Online Signing.

- On receiving $(\mathsf{sign}, sid||\mu, \{m^{(\nu)}\}_{\nu\in[\ell]})$ from some party $P_i, i \in S$, if $(\mathsf{offsigned}, sid)$ exists in memory, $sid||\mu$ is fresh and $\mu \in [\hat{\ell}]$, send $(\mathsf{sign}, sid||\mu, \{m^{(\nu)}\}_{\nu\in[\ell]}, i)$ to Sim.
- After receiving $(\mathsf{sign}, sid||\mu, \{m^{(\nu)}\}_{\nu\in[\ell]})$ from all parties in the signer set S, retrieve $k^{(\nu)} = k^{(\mu\nu)}$ and compute $r^{(\nu)} = \mathsf{F}(R^{(\mu\nu)})$, $s^{(\nu)} = \dfrac{m^{(\nu)} + r^{(\nu)}x}{k^{(\nu)}} \bmod q$, and set $s^{(\nu)} = \mathsf{min}(s^{(\nu)}, q - s^{(\nu)})$ for each $\nu \in [\ell]$.
- Send $\{(r^{(\nu)}, s^{(\nu)})\}_{\nu\in[\ell]}$ to Sim. If receiving $(\mathsf{exposed}, sid, j)$ from Sim, remove party P_j from the signer set S.
- If $|S| \geq t + \ell$, send $\{(r^{(\nu)}, s^{(\nu)})\}_{\nu\in[\ell]}$ to all signers in S and store $(\mathsf{signed}, sid||\mu)$ in memory.

The proof of our main security theorem (Theorem 1) for our threshold ECDSA scheme is as follows.

Proof. Let $\mathcal{P}^* \subseteq [n]$ be the set of the parties corrupted by the adversary, where $|\mathcal{P}^*| \leq t$ and let $\mathcal{H} = [n] \setminus \mathcal{P}^*$ be the set of honest parties. We construct a simulator Sim that interacts with $\mathcal{F}_{\mathsf{ECDSA}}$ and the adversary $\mathcal{A}$, such that $\mathcal{A}$'s view in the interaction with Sim is indistinguishable from its view in the real execution of the protocol. Sim acts as follows.

Key Generation.

1. On receiving $(\mathsf{keygen}, sid, i)$ for some $i \in \mathcal{H}$ from $\mathcal{F}_{\mathsf{ECDSA}}$, reply (keygen, sid) on behalf of $P_j, j \in \mathcal{P}^*$. Let $Q = [n]$.
2. On receiving $(ek_j, \pi_{\mathsf{key}_j})$ from $\mathcal{A}$, if π_{key_j} is invalid, sends $(\mathsf{exposed}, sid, j)$ to $\mathcal{F}_{\mathsf{ECDSA}}$ and sets $Q = Q \setminus \{j\}, \mathcal{P}^* = \mathcal{P}^* \setminus \{j\}$. Otherwise, Sim extracts dk_j from the proofs π_{key_j} for $j \in \mathcal{P}^*$, using the fact that in PVSSeq, these are proofs of knowledge.
3. On behalf of each honest party $i \in \mathcal{H}$, Sim generates and publishes the first-round messages $(\{\mathcal{P}_{ik}, \mathcal{C}_{ik}\}_{k \in Q}, \pi_{\mathsf{share}_i})$ as specified in the protocol description. Specifically, it samples a random $w_i \leftarrow\!\!\$\ \mathbb{Z}_q$ and takes it as input to run $\mathsf{PVSSeq.Share}(\cdot)$. Sim stores the shares w_{ij} for $j \in \mathcal{P}^*$.
4. On receiving $\{(\{\mathcal{P}_{jk}, \mathcal{C}_{jk}\}_{k \in Q}, \pi_{\mathsf{share}_j})\}_{j \in \mathcal{P}^*}$ from $\mathcal{A}$,
   - If π_{share_j} is not valid, Sim sends $(\mathsf{exposed}, sid, j)$ to $\mathcal{F}_{\mathsf{ECDSA}}$. Let $Q = Q \setminus \{j\}, \mathcal{P}^* = \mathcal{P}^* \setminus \{j\}$
   - Sim uses dk_k to decrypt every $\mathcal{C}_{kj}, \forall k, j \in \mathcal{P}^*$, getting w_{kj}. Then, Sim gets a knowledge of $x_j = \sum_{k \in Q} w_{kj}$.
   - With X received from $\mathcal{F}_{\mathsf{ECDSA}}$, build a degree-$(t+\ell-1)$ polynomial $f_X(\cdot)$ over $\mathbb{G}$ s.t. $f_X(-\nu+1) = X, \forall \nu \in [\ell]$ and $f_X(j) = x_j G, \forall j \in \mathcal{P}^*$. For each $i \in \mathcal{H}$, set $X_i = f_X(i) \in \mathbb{G}$ and simulate π_{dld_i}.
   - Sim invokes VNIM's simulating algorithm $\mathsf{SimInput}_B(\mathsf{crs}, X_i) \to (\mathsf{ec}_{Bx_i}, \pi_{Bx_i}, \mathsf{st}^*_{x_i})$, as defined in the privacy definition of VNIM (Section 3.1).
   - Broadcast $(X_i, \pi_{\mathsf{dld}_i}, \mathsf{ec}_{Bx_i}, \pi_{Bx_i})$ on behalf of honest P_i.
5. On receiving $\{(X_j, \pi_{\mathsf{dld}_j}, \mathsf{ec}_{Bx_j}, \pi_{Bx_j}\}_{j \in \mathcal{P}^*}$ from $\mathcal{A}$, if π_{Bx_j} or π_{dld_j} is not valid, Sim send $(\mathsf{exposed}, sid, j)$ to $\mathcal{F}_{\mathsf{ECDSA}}$. Let $Q = Q \setminus \{j\}, \mathcal{P}^* = \mathcal{P}^* \setminus \{j\}$.

Analysis. We show that the execution with Sim and $\mathcal{F}_{\mathsf{ECDSA}}$'s key generation phase is indistinguishable from an execution of the real DKG protocol. First, we notice that the encryption keys ek_j and encrypted shares $\mathcal{C}_{jk}$ generated by $\mathcal{A}$ are valid CL public keys and ciphertexts if the corresponding zero-knowledge proofs are valid, due to the verifiability of PVSSeq. And, all messages produced by Sim in the first round are exactly as in the protocol, and Q is exactly as it would be in real protocols. The values $x_j,\ j \in \mathcal{P}^*$ computed by Sim are exactly the same as the adversary would obtain in the protocol. For each honest party's sharing of $(t+\ell-1)$-degree $f_i(\cdot)$, the adversary can get at most t evaluations $\{f_i(j)\}_{j \in \mathcal{P}^*}$. Then, by properties of interpolation, there exist q polynomials $f^*(\cdot)$ s.t. $f^*(j) = f_i(j),\ \forall j \in \mathcal{P}^*$. Thus, the adversary has no information about honest parties' share $f_i(j),\ j \in \mathcal{H}$. Moreover, due to the privacy of PVSSeq, $(\{\mathcal{P}_{ik}, \mathcal{C}_{ik}\}_{k \in Q}, \pi_{\mathsf{share}_i})$ does not reveal more information beyond that.

The first main difference between the real and simulated protocols is that Sim generates honest parties' X_i based on X and the adversary's shares x_j, and simulates the corresponding zero-knowledge proofs. From the adversary's perspective, due to the zero-knowledge property of π_{dld_i}, it has no additional information about x_i beyond the fact that $x_i G = X_i$ and $x_i = f(i)$ for a uniformly random secret polynomial $f(\cdot)$ conditioned to $f(j) = x_j$ for $j \in \mathcal{P}^*$. Thus, the real x_i is exactly the same distribution used by the functionality to choose the ECDSA public key $X = xG$, where $x \leftarrow\!\!\$\ \mathbb{Z}_q$. Due to the soundness of π_{dld_j}, X_j revealed by $\mathcal{A}$ should be equal to $x_j G$ when π_{dld_j} is valid.

The second main difference is that the simulator Sim invokes the simulation algorithm of the VNIM scheme to simulate the encodings of x_i. By the privacy property of VNIM, these simulated encodings are computationally indistinguishable from those in the real execution. At the same time, due to the soundness of π_{Bx_j}, $\mathcal{A}$'s ec_{Bx_j} is x_j's encoding when π_{Bx_j} is valid.

Offline Signing. With the signer set S, let $\mathcal{P}^*_S = \mathcal{P}^* \cap S$ and $\mathcal{H}_S = \mathcal{H} \cap S$.

1. On receiving $(\mathsf{offsign}, sid, i)$ for some $i \in \mathcal{H}_S$ from $\mathcal{F}_{\mathsf{ECDSA}}$, reply $(\mathsf{offsign}, sid)$ on behalf of $P_j, j \in \mathcal{P}^*_S$.
2. On receiving $\{R^{(\mu\nu)}\}_{\mu \in [\hat{\ell}], \nu \in [\ell]}$ from $\mathcal{F}_{\mathsf{ECDSA}}$, Sim generate the first-round messages $(\{\mathcal{C}_{ia}\}_{a \in S}, \pi_{\mathsf{Sh}_i})$ as specified in the protocol description, for each $i \in \mathcal{H}_S$. On receiving $\{(\{\mathcal{C}_{ja}\}_{a \in S}, \pi_{\mathsf{Sh}_j})\}_{j \in \mathcal{P}^*_S}$ from $\mathcal{A}$,
   - If π_{Sh_j} is not valid, send $(\mathsf{exposed}, sid, j)$ to $\mathcal{F}_{\mathsf{ECDSA}}$ and set $S = S \setminus \{j\}, \mathcal{P}^*_S = \mathcal{P}^*_S \setminus \{j\}$. Otherwise, decrypt each $\mathcal{C}_{aj}$ to get $k_{aj}, a, j \in \mathcal{P}^*_S$. By combining the honest parties' shares, Sim can get $\{k_{aj}\}_{a \in S}$. With $\boldsymbol{\Psi} = [\psi_a^{(\mu)}]_{a \in S}^{\mu \in [\hat{\ell}]}$, Sim can get $k_j^{(\mu)} = \sum_{a \in S} \psi_a^{(\mu)} k_{aj}$ for each $\mu \in [\hat{\ell}]$.

- For each $\mu \in [\hat{\ell}]$, build a degree-$(t+\ell-1)$ polynomial $f_R^{(\mu)}(\cdot)$ over $\mathbb{G}$ s.t. $f_R^{(\mu)}(-\nu+1) = R^{(\mu\nu)}, \nu \in [\ell]$ and $f_R^{(\mu)}(j) = k_j^{(\mu)} G$ for each $j \in \mathcal{P}_S^*$.
- For each $i \in \mathcal{H}_S$, set $R_i^{(\mu)} = f_R^{(\mu)}(i) \in \mathbb{G}$ and simulate $\tau_{\mathsf{dld}_i}^{(\mu)}$. Broadcast them on behalf of honest P_i. On receiving $\mathcal{A}$'s $(\{R_j^{(\mu)}\}_{\mu\in[\hat{\ell}]}, \tau_{\mathsf{dld}_j})$, if τ_{dld_j} is not valid, send $(\mathsf{exposed}, sid, j)$ to $\mathcal{F}_{\mathsf{ECDSA}}$ and set $S = S \setminus \{j\}, \mathcal{P}_S^* = \mathcal{P}_S^* \setminus \{j\}$.

3. Sim simulates the VNIM-based executions for $\mathcal{A}$. For each $\mu \in [\hat{\ell}]$, let $R_i = R_i^{(\mu)}$ and do the following.
   - On behalf of each $i \in \mathcal{H}_S$, pick $\Gamma_i \leftarrow\!\!\$\ \mathbb{G}$ and invoke VNIM's simulating algorithm $\mathsf{SimInput}_A(\mathsf{crs}, \Gamma_i) \to (\mathsf{ec}_{A\gamma_i}, \pi_{A\gamma_i}, \mathsf{st}_{\gamma_i}^*)$ as a role of A. Broadcast $(\Gamma_i, \mathsf{ec}_{A\gamma_i}, \pi_{A\gamma_i})$ on behalf of honest party P_i.
   - On receiving $(\Gamma_j, \mathsf{ec}_{A\gamma_j}, \pi_{A\gamma_j})$ from $\mathcal{A}$, if $\pi_{A\gamma_j}$ is invalid, send $(\mathsf{exposed}, sid, j)$ to $\mathcal{F}_{\mathsf{ECDSA}}$ and set $S = S \setminus \{j\}, \mathcal{P}_S^* = \mathcal{P}_S^* \setminus \{j\}$.
   - Invoke $\mathsf{SimInput}_B(\mathsf{crs}, R_i) \to (\mathsf{ec}_{Bk_i}, \pi_{Bk_i}, \mathsf{st}_{k_i}^*)$. For each $j \in S$, $\mathsf{SimOutput}_B(\mathsf{ec}_{A\gamma_j}, \pi_{A\gamma_j}, \mathsf{st}_{k_i}^*, R_i) \to (B_{ij}, -)$ $\mathsf{SimOutput}_B(\mathsf{ec}_{A\gamma_j}, \pi_{A\gamma_j}, \mathsf{st}_{x_i}^*, X_i) \to (\hat{B}_{ij}, -)$. During the above simulation, Sim can get $\mathcal{A}$'s shares $\{\gamma_j, \alpha_{ji}, \hat{\alpha}_{ji}\}_{j\in\mathcal{P}_S^*, i\in\mathcal{H}_S}$ (c.f. Appendix B.4). Simulate the proofs $\tau_i, \hat{\tau}_i$.
   - Broadcast $(\mathsf{ec}_{Bk_i}, \pi_{Bk_i}, \{B_{ij}, \hat{B}_{ij}\}_{j\in S}, \tau_i, \hat{\tau}_i)$ on behalf of each honest party P_i.
   - On receiving $\mathcal{A}$'s $(\mathsf{ec}_{Bk_j}, \pi_{Bk_j}, \{B_{ja}, \hat{B}_{ja}\}_{a\in S}, \tau_j, \hat{\tau}_j)$, if π_{Bk_j}, τ_j or $\hat{\tau}_j$ is not valid, send $(\mathsf{exposed}, sid, j)$ to $\mathcal{F}_{\mathsf{ECDSA}}$ and set $S = S \setminus \{j\}, \mathcal{P}_S^* = \mathcal{P}_S^* \setminus \{j\}$.
   - Parse $\mathsf{st}_{\gamma_i}^* = (v_i, v_i')$ where $v_i \leftarrow\!\!\$\ \mathbb{Z}_q$ is a random value that replaces the real γ_i and v_i' is the randomness used in $\mathsf{ec}_{A\gamma_i}$ (c.f. Appendix B.4). Invoke $\mathsf{Decode}_A(\mathsf{crs}, \mathsf{ec}_{Bk_j}, v_i', v_i) \to \alpha_{ij}$ and $\mathsf{Decode}_A(\mathsf{crs}, \mathsf{ec}_{Bx_j}, v_i', v_i) \to \hat{\alpha}_{ij}$ for each $i \in \mathcal{H}_S, j \in \mathcal{P}_S^*$.

Analysis. The simulation of the $R^{(\mu\nu)}$-generation phase is similar to that of the DKG phase, with the sole difference being the incorporation of the hyper-invertible matrix $\boldsymbol{\Psi}$. First, the values $k_j^{(\mu)}$ for $j \in \mathcal{P}_S^*$ computed by Sim are identical to those the adversary would obtain in the real protocol execution. Simultaneously, Sim generates the honest parties' $R_i^{(\mu)}$ based on $R^{(\mu\nu)}$ and the adversary's shares $k_j^{(\mu)}$ for each $\mu \in [\hat{\ell}]$. That is, Sim builds $\hat{\ell}$ $(t+\ell-1)$-degree polynomials $f_R^{(\mu)}(\cdot)$ s.t. $f_R^{(\mu)}(-\nu+1) = R^{(\mu\nu)}, \nu \in [\ell]$ and $f_R^{(\mu)}(j) = k_j^{(\mu)} G$ for each $j \in \mathcal{P}_S^*$. In the real protocol, due to the hyper-invertibility of matrix $\boldsymbol{\Psi}$, the aggregated $\hat{\ell}$ polynomials (of degree $t+\ell-1$) are all uniformly random, and the adversary learns only t evaluation points among them. This yields the exact same distribution as when the ideal functionality uniformly samples $R^{(\mu\nu)}$ at random and then Sim builds corresponding polynomials $f_R^{(\mu)}(\cdot)$ as above.

Due to the privacy property of VNIM, the transcripts generated by Sim through invoking $\mathsf{SimInput}_A(\cdot)$, $\mathsf{SimInput}_B(\cdot)$, and $\mathsf{SimOutput}_B(\cdot)$ are indistinguishable from those produced in the real protocol execution. And, due to the verifiability of VNIM, the encodings $\mathsf{ec}_{A\gamma_j}, \mathsf{ec}_{Bk_j}$ and $\{B_{ja}, \hat{B}_{ja}\}_{a\in S}$ produced by the adversary are correct, provided that the associated zero-knowledge proofs are valid. Since Sim honestly verifies the adversary's zero-knowledge proofs and exposes any malicious parties, the set S is identical to what it would be in the real execution.

Online Signing.

1. Pick $\bar{x} \leftarrow\!\!\$\ \mathbb{Z}_q$ and regard $\bar{x}$ as the dummy secret key of ECDSA. With the knowledge of $\{x_j\}_{j\in\mathcal{P}_S^*}$, build a $(t+\ell-1)$-degree polynomial $f_x(\cdot)$ over $\mathbb{Z}_q$ s.t.
   - $f_x(0) = f_x(-1) = \cdots = f_x(-\ell+1) = \bar{x}$ and
   - $f_x(j) = x_j$ for each $j \in \mathcal{P}_S^*$.
2. On receiving $(\mathsf{sign}, sid||\mu, \{m^{(\nu)}\}_{\nu\in[\ell]}, i)$ from some $i \in \mathcal{H}_S$, reply $(\mathsf{sign}, sid||\mu, m)$ on behalf of $P_j, j \in \mathcal{P}_S^*$.
3. On receiving $\{(r^{(\nu)}, s^{(\nu)})\}_{\nu\in[\ell]}$ from $\mathcal{F}_{\mathsf{ECDSA}}$ where $r^{(\nu)} = \mathsf{F}(R^{(\mu\nu)})$ for current index μ, Sim interpolates the polynomial $p(\cdot)$ with $p(-\nu+1) = m^{(\nu)}(r^{(\nu)})^{-1}, \forall \nu \in [\ell]$.
4. Compute $\bar{k}^{(\nu)} = \dfrac{m^{(\nu)} + r^{(\nu)}\bar{x}}{s^{(\nu)}}, \forall \nu \in [\ell]$ and regard $\bar{k}^{(\nu)}$ as the ν-th dummy secret nonce used for the ν-th signature. With the knowledge of $\{k_j = k_j^{(\mu)}\}_{j\in\mathcal{P}_S^*}$ for current index μ, build a degree-$(t+\ell-1)$ polynomial $f_k(\cdot)$ over $\mathbb{Z}_q$ s.t. $f_k(-\nu+1) = \bar{k}^{(\nu)}, \forall \nu \in [\ell]$ and $f_k(j) = k_j, \forall j \in \mathcal{P}_S^*$.
5. For each honest party $P_i, i \in \mathcal{H}_S$, simulate shares $\bar{x}_i = f_x(i), \bar{k}_i = f_k(i)$. With the knowledge of $\gamma_j, \alpha_{ji}, \hat{\alpha}_{ji}$, simulate honest parties' shares $\delta_{ij} = \gamma_j \bar{k}_i - \alpha_{ji} + \alpha_{ij}$, $\hat{\delta}_{ij} = \gamma_j \bar{x}_i - \hat{\alpha}_{ji} + \hat{\alpha}_{ij} \bmod q, j \in \mathcal{P}_S^*$.

6. For each honest P_i, using $(v_i, \bar{x}_i, \bar{k}_i, \{\delta_{ij}, \hat{\delta}_{ij}\}_{j\in\mathcal{P}^*_S})$ to simulate:
   - for $j \in \mathcal{H}_S$, $a_{ij} = v_i\bar{k}_j + z(j)$, $b_{ij} = v_i(p(j) + \bar{x}_j) + \hat{z}(j)$
   - for $j \in \mathcal{P}^*_S$, $a_{ij} = \delta_{ij} + z(j)$, $b_{ij} = v_i p(j) + \hat{\delta}_{ij} + \hat{z}(j)$

   where $z(\cdot), \hat{z}(\cdot)$ are generated in the same way as the real execution. For each $\nu \in [\ell]$, compute $A_i^{(\nu)} \coloneqq \sum_{j\in S} \lambda_j^{(\nu)}(a_{ij}G + B_{ji} - B_{ij})$, $B_i^{(\nu)} \coloneqq \sum_{j\in S} \lambda_j^{(\nu)}(b_{ij}G + \hat{B}_{ji} - \hat{B}_{ij})$, and simulate the proof π'_i for the statements $\{R^{(\nu)}, A_i^{(\nu)}, C^{(\nu)}, B_i^{(\nu)}\}_{\nu\in[\ell]}$.
7. Boradcast $(\{a_{ij}, b_{ij}\}_{j\in S}, \pi'_i)$ to $\mathcal{A}$ on behalf of each honest party P_i. After receiving $\mathcal{A}$'s $(\{a_{ji}, b_{ji}\}_{i\in S}, \pi'_j)$, if π'_j is not valid, send $(\mathsf{exposed}, sid, j)$ to $\mathcal{F}_{\mathsf{ECDSA}}$.

Analysis. We first emphasize that $\{\alpha_{ji}, \hat{\alpha}_{ji}\}_{j\in\mathcal{P}^*_S, i\in\mathcal{H}_S}$ obtained by Sim are identical to those the adversary would receive in the real execution (c.f. Appendix B.4). Furthermore, $\{\alpha_{ij}, \hat{\alpha}_{ij}\}_{j\in\mathcal{P}^*_S, i\in\mathcal{H}_S}$ computed by Sim satisfy $\alpha_{ij} + \beta_{ji} = v_i k_j$ and $\hat{\alpha}_{ij} + \hat{\beta}_{ji} = v_i x_j$ where $\{\beta_{ji}, \hat{\beta}_{ji}\}_{j\in\mathcal{P}^*_S, i\in\mathcal{H}_S}$ are the shares adversary would receive in the execution of VNIM. At the same time, if $\mathcal{A}$'s π'_j is accepting, the corresponding $\{a_{ji}, b_{ji}\}_{j\in\mathcal{P}^*_S, i\in S}$ are honestly evaluated from the additive shares derived via VNIM, based on the soundness of π'_j, the correctness of $B_{ij}, \hat{B}_{ij}$, and Lemma 5. After discarding the invalid a_{ij}, b_{ij} according to π'_i and updating S, the simulation in Step 5&6 will make the final assembly generate

$$r^{(\nu)} \cdot \frac{\sum_{i,j\in S} \lambda_i^{(\nu)}\lambda_j^{(\nu)} b_{ij}}{\sum_{i,j\in S} \lambda_i^{(\nu)}\lambda_j^{(\nu)} a_{ij}} = \frac{m^{(\nu)} + r^{(\nu)}\bar{x}}{\bar{k}^{(\nu)}} = s^{(\nu)},$$

with $|S| \geq t + \ell$. If $\mathcal{A}$ cheats on their shares, the offset caused on $s^{(\nu)}$ will be the same as that in the real protocol.

The simulated $\{a_{ij}, b_{ij}\}_{j\in S}$ have the same distribution as those in the real protocol due to the presence of random $v_i \leftarrow\$ \mathbb{Z}_q$ and masking polynomials $z(\cdot), \hat{z}(\cdot)$. Although dummy shares $\{\bar{x}_i, \bar{k}_i\}_{i\in\mathcal{H}_S}$ are utilized in the simulation, causing $\{R^{(\nu)}, A_i^{(\nu)}, C^{(\nu)}, B_i^{(\nu)}\}_{\nu\in[\ell]}$ to deviate from DDH tuples, the simulated proof π'_i generated by Sim remains indistinguishable from that in the real execution, owing to the zero-knowledge property of the proof system. □