Bandwidth-Efficient Robust Threshold ECDSA in Three Rounds

Yingjie Lyu
Shandong University
Qingdao, China
yingjielyu@mail.sdu.edu.cn

Zengpeng Li*
Shandong University
Qingdao, China
zengpengliz@gmail.com

Hong-Sheng Zhou
Virginia Commonwealth University
USA
hongsheng.zhou@gmail.com

Haiyang Xue
Singapore Management University
Singapore
haiyangxc@gmail.com

Mei Wang
Shandong University
Qingdao, China
wangmeiz@sdu.edu.cn

Shuchao Wang
Shandong University
Qingdao, China
wangshuchao.shaw@gmail.com

Mengling Liu
The Hong Kong Polytechnic University
Hong Kong
mengling.liu@connect.polyu.hk

ABSTRACT

Threshold ECDSA schemes distribute the capability of issuing signatures to multiple parties. They have been used in practical MPC wallets holding cryptocurrencies. However, most prior protocols are not robust, wherein even one misbehaving or non-responsive party would mandate an abort. Robust schemes have been proposed (Wong et al., NDSS '23, '24), but they do not match state-of-the-art number of rounds which is only three (Doerner et al., S&P '24).

In this work, we propose robust threshold ECDSA schemes RompSig-Q and RompSig-L that each take three rounds (two of which are broadcasts). Building on the works of Wong et al. and further optimized towards saving bandwidth, they respectively take each signer $(1.0t + 1.6)$ KiB and 3.0 KiB outbound broadcast communication, and thus exhibit bandwidth efficiency that is competitive in practical scenarios where broadcasts are natively handled. RompSig-Q preprocesses multiplications and features fast online signing; RompSig-L leverages threshold CL encryption for scalability and dynamic participation.

1 INTRODUCTION

In a (t, n) threshold signature scheme [30], the signing key is secret-shared among n parties, such that generating a valid signature for a given message requires a quorum of at least t signers to collaboratively execute a multi-party signing protocol. Threshold signature schemes enhance security by removing the *single point of failure*, requiring an adversary to corrupt at least t parties to forge a signature, thereby significantly increasing the difficulty of an attack. Also, through a process known as a *key refresh*, some leaked or lost secret key shares can be invalidated and renewed. In recent years, there has been a particular research interest in efficient threshold signature schemes compatible with the Elliptic Curve Digital Signature Algorithm (ECDSA), mainly because of their application to highly-secure cryptocurrency wallets.

ECDSA features a threshold-unfriendly, non-linear structure compared to Schnorr or BLS. Signing a message $\texttt{msg}$ using the secret key $x \in \mathbb{Z}_q$ outputs (r, s), where r is the x-coordinate of the elliptic curve point $R = g^k$ for a uniformly random nonce $k \in \mathbb{Z}_q$, and $s = k^{-1}(\mathsf{H}(\texttt{msg}) + rx) \bmod q$, where H is SHA-2 mapping the message to an integer in $\mathbb{Z}_q$. In the multi-party case, both x and k must be secret-shared. Many tools have been used for the secure computation of s, including Linearly-Homomorphic Encryption (LHE) schemes of Paillier [40, 16], Castagnos–Laguillaumie (CL) [18, 19, 65, 64], and Joye–Libert [66]; Oblivious Transfer (OT) [56, 34, 35, 25]; and also Pseudorandom Correlated Generator (PCG) [1].

Since *guaranteed output delivery* is generally impossible in the dishonest-majority case, most prior works only aim for the minimal guarantee of *security with abort* [40, 56, 18, 35, 1]. In these protocols, any misbehaving or non-responsive party would irrevocably cause the signing session to abort, and there is no way to reliably determine the party responsible for an abort. This can lead to Denial of Service (DoS) attacks because if malicious players persist, signing may continue to fail. Building on them, some later works implement *identifiable abort* (IA) [16, 19, 25] as a DoS mitigation: If signing fails, at least one cheater can be publicly identified, and subsequent reruns can exclude the culprit.

While schemes with IA are often seen as good enough, it is not the best guarantee possible. The well-known threshold BLS scheme of Boldyreva [8] satisfies *t-robustness*, i.e., if t parties honestly participate in the signing protocol, then it always outputs a valid signature, even if the adversary corrupting up to $t - 1$ parties actively seeks to disrupt it. This does not contradict the impossibility of guaranteed output delivery under a dishonest majority as the condition may not hold at times, but proves to be a useful desideratum, allowing the upper-level system to include a few redundant signers so that an equivalent number of nodes can fail without disrupting threshold signing.

There are several real-world scenarios wherein robustness is relevant. The first class is time-critical systems. Using a robust scheme with some redundant nodes, timeouts due to a few nodes failing can often be avoided. The second class is large-scale deployments over the internet. Blockchain applications holding public funds such as decentralized exchanges and cross-chain links may need threshold ECDSA with large quorums to ground trust in decentralization. Moreover, robust signing means the online-phase players do not

*Corresponding author

Table 1: Comparison of main threshold ECDSA schemes.

Scheme	Rounds		Guarantees	Main Tool
	offline	online		
GG18 [40]	3	5	–	Paillier
LN18 [56]	4	1	–	Paillier/OT
CGGMP20 [16]	6/3	1	IA	Paillier
ANOSS22 [1]	PCG	2	–	PCG
CCLST23 [19]	6	1	IA	CL
WMYC23 [65]	6	1	IA, Robust	CL
DKLs24 [35]	2	1	–	OT
WMC24 [64]	3	1	IA, Robust	T-CL
CDKS24 [25]	6	1	IA	OT
RompSig-Q	2	1	IA, Robust	CL
RompSig-L	2	1	IA, Robust	T-CL

IA: Identifiable Abort. All schemes with IA and/or robustness require broadcasts. Offline rounds are those independent of messages; online rounds happen after the message to be signed is known. Baseline guarantee (–) is *security with abort*. OT: Oblivious Transfer. PCG: Pseudorandom Correlation Generator. CL: Castagnos–Laguillaumie encryption. T-CL: Threshold CL.

have to be exactly the ones that perform preprocessing, and this decoupling simplifies system design.

Among existing threshold-optimal ECDSA schemes, i.e., those supporting arbitrary choice of the threshold t, the only t-robust ones are WMYC23 [65] and WMC24 [64]. Interestingly, WMYC23 focuses on low online latency, and WMC24 focuses on scalability, with different performance tradeoffs. However, they respectively require 7 and 4 rounds of interaction; all offline rounds are broadcasts. In contrast, the state-of-the-art scheme of Doerner et al. [35] only requires three peer-to-peer rounds. In peer-to-peer networks, each broadcast round takes multiple peer-to-peer rounds, and incurs a significant overhead. Schemes such as Canetti et al. [16] propose using slightly cheaper echo-broadcasts for the broadcast rounds when cheater identifiability (and robustness) are not required. While broadcasts and consensus are necessary in the robust setting [26] and the overhead cannot be entirely removed, reducing the number of broadcast rounds is practically meaningful. Thus, we ask the following question:

> *Is it possible to construct robust threshold ECDSA with fewer broadcast rounds?*

1.1 Our Results

In this work, we provide an affirmative answer to the above question. We construct two robust threshold ECDSA schemes, RompSig-Q and RompSig-L, that each only require two rounds of broadcasts in the offline phase and a non-interactive online phase. We compare RompSig-Q and RompSig-L with major existing works in Table 1.

1.1.1 Overview of RompSig-Q. The starting point of RompSig-Q is the WMYC23 [65] scheme. Their main novelty is using a Distributed Key Generation (DKG) subprotocol in signing to generate (t, n) Shamir secret-shared nonces, instead of (t, t) additively shared as in normal non-robust protocols, so that signing can tolerate parties dropping out. Their protocol uses the Multiplication-to-Addition (MtA) technique, which originates from [40, 56], allows low online-phase latency, and has been used by most protocols ever since. The major problem is that their protocol takes a total of 7 rounds.

Our first observation is that WMYC23 uses a less round-efficient high-level framework than DKLs24 [35]. The lowest possible number of rounds using the same framework is four [16]. Applying the new framework of DKLs24 only saves one round, though.

Another observation is that WMYC23 uses a three-round DKG protocol similar to [43] to generate nonces, in which one round is for complaining against senders of inconsistent shares. Only afterwards are the encryptions of nonces prepared for the MtA conversion. Moreover, misbehavior in the MtA conversion is also identified in an extra round of interactive complaints.

We propose using the non-interactive DKG scheme of Kate et al. [49] to generate Shamir secret-shared nonces and also ciphertexts suitable for MtA, in only one round. This replaces the first three rounds of DKG and ciphertext generation in WMYC23. A technicality arises: if done naively, this would enable the adversary to bias the distribution of nonces. We use a technique of alternating group generators proposed by Neji et al. [58] as a countermeasure. The opening of $R = g^k$ is delayed to the second round, and the uniform distribution of it is ensured by DDH. We also eliminate the complaining round after the MtA conversion in WMYC23, replacing it with a publicly-verifiable, non-interactive zero-knowledge (NIZK) proof of correct behavior. This finally brings the number of broadcast rounds in the offline phase to two.

Replacing interactive complaints and proving honest behavior in NIZK incurs $O(n^2)$ computational cost for NIZK verification. Indeed, our RompSig-Q is heavier in computation than WMYC23. Nevertheless, for small quorum sizes common for cryptocurrency wallets, the computational performance is still practical: ~700 milliseconds for 3 parties and ~1300 milliseconds for 5 parties in single-threaded CPU time, mostly spent in the offline phase (the online phase takes less than 1 millisecond).

We use a few techniques to reduce the sizes of protocol messages. These are well-known in the literature but have not often been used for threshold ECDSA. First, we observe that the MtA conversion can work on vector (multi-receiver) ciphertexts [6], where the encryption randomness is reused, which are output by joint PVSS. CL ciphertexts take a significant proportion of the total communication, and using multi-receiver ciphertexts throughout signing halves such costs. Second, we use the batching technique for Fiat-Shamir NIZKs proposed by Gennaro et al. [44], compressing proofs for multiple MtA instances into one, the size of which only increasing logarithmically with the number of parties. Finally, we use the alternative Fiat-Shamir formulation of the original Schnorr signature scheme [60]: instead of taking the first and third moves in a Sigma protocol as the NIZK argument, we take the second and third, omitting the first-move commitment that contains rather big class group elements. The resulting scheme takes each party $(1.0t + 1.6)$ KiB broadcast communication in a signing session.

We remark that in peer-to-peer networks the actual communication will be much higher due to the broadcast overhead, while schemes such as DKLs24 do not need broadcasts. However, there

exist schemes that communicate exclusively over broadcast channels to achieve identifiable abort (or robustness) [16, 65, 64, 25], and our communication cost is quite competitive in this realm.

1.1.2 Overview of RompSig-L. The construction of our second scheme RompSig-L is more straightforward. Braun et al. [14] proposed a threshold version of the CL linearly-homomorphic encryption scheme, and suggested some improvements in [13]. Building on threshold CL encryption, Wong et al. [64] revived the first attempt of Gennaro et al. [41] at threshold-optimal ECDSA. In this line of design, an encrypted signature is incrementally assembled by the parties and finally opened via threshold decryption. The six-round protocol of Gennaro et al. [41] builds on threshold Paillier encryption, which is impractical because the distributed setup of it involves sampling an RSA modulus with factorization unknown by any party, and this is very expensive and takes about 10 minutes [31]. This was the main reason they moved to a design using the MtA technique [40]. On the other hand, threshold CL encryption has transparent setup and efficient DKG, and is therefore practically efficient. Applying the updated three-round framework of DKLS24 to WMC24 suffices to give a practical three-round design.

The resulting scheme RompSig-L features a constant broadcast communication of 3 KiB, and the computation is much more scalable than RompSig-Q. However, as is the case with WMC24, its online phase is heavier in computation, as threshold decryption requires multiple class-group exponentiations.

Another special feature of RompSig-L, inherited from [64], is support for dynamic participation. Assuming a public bulletin board, for signing to proceed, it is sufficient that each round is joined by t provably honest players; a player does not have to be in the session from the start, and can leave any time. This is desirable if the parties are ephemeral non-coordinated Internet nodes.

1.2 Related Work

The most notable works on threshold-optimal ECDSA are [40, 56, 16, 19, 65, 64, 1, 35]. We also list [41, 9, 33, 18, 39, 29, 68, 7, 11, 66]. The works of Gennaro and Goldfeder [40] and Lindell and Nof [56] jointly set the standard of using the Bar-Ilan–Beaver inversion trick [5] and the MtA conversion. The basic techniques are explained in a 2020 review [4]. Doerner et al. [35] also gives a very comprehensive review of prior works.

Reducing the number of rounds is important from both theoretical and practical aspects. Until recently, the best was [16], with 3-round presigning and non-interactive online signing. Based on a different structural idea called *the ECDSA tuple* from [1], Doerner et al. [35] first achieved 3-round signing. The schemes of Doerner et al. [34, 35] are computationally very efficient because they use OT extension for 2-party multiplication, which mostly calls fast symmetric primitives. However, these schemes incur high bandwidth usage. Abram et al. [1] use a Pseudorandom Correlated Generator (PCG) to achieve extremely low amortized bandwidth cost, but their scheme only supports full threshold, i.e., $t = n$, hence is not threshold-optimal.

Schemes based on LHE are the most studied. An attack [63] declares Paillier-based schemes [40, 16] insecure without range proofs, which are costly in both computation and communication. Secure without range proofs, CL-based schemes [18, 19, 29, 68, 65, 64] are on the rise. CL has much shorter ciphertexts than Paillier at the same level of security. Notably, an optimized implementation [12] has improved its speed to surpass Paillier.

Some works consider robustness but only in the online phase; their presigning phases cannot tolerate faults as in the non-robust schemes [39, 59]. The protocol of Groth and Shoup [46] is robust but not threshold optimal, supporting only $t < n/2$. The protocol of Zyskind et al. [69] achieves robustness assuming external mechanisms, namely *confidential smart contracts*, holding half of secret key shares and being semi-honest.

Wong et al. [65] are the first to construct fully robust threshold-optimal ECDSA. They use Distributed Key Generation [43] to generate Shamir secret-shared nonces, and then apply MtA preprocessing. In a follow-up work, Wong et al. [64] instead build on threshold CL, and achieve robustness under dynamic participation, which resembles YOSO [45] and Fluid MPC [24] models. Both schemes assume broadcasts, which is necessary for robustness or identifiable abort [26]. These schemes take 7 and 4 rounds, respectively.

Other schemes with constant broadcast communication are [41, 9, 38], but they rely on threshold Paillier encryption, which is impractical without a trusted setup as distributed generation of Paillier secret keys involves sampling a RSA modulus and is computationally very expensive. This is indeed why [40] later appeared.

Some threshold ECDSA schemes require an honest majority [42, 27, 59, 46, 51] and do not work for the case of $t > n/2$. They typically require at least $2t - 1$ players to sign. This can simplify design but does not allow flexibility in the choice of threshold.

Earlier two-party ECDSA schemes [54, 57, 55, 33, 20] pave the way towards the multi-party, dishonest-majority case. Some revisit the two-party case later [67, 52, 62, 10, 53, 2].

1.3 Technical Overview

Recall that the task of threshold ECDSA is to compute $R = g^k$, $r = R|_{x\text{-axis}}$, and $s = k^{-1}(m + rx) \bmod q$ for $m = \mathsf{H}(\mathtt{msg})$, from the secret-shared signing key x and random nonce k. At the crux of this task is the challenge posed by the non-linearity of $k^{-1} \bmod q$ multiplied with x. The structural approach we take is in line with [56, 1, 35], termed *the ECDSA tuple*: With a secret-shared mask ϕ, the final result can be rewritten as $s = (m\phi + rx\phi)/k\phi$.

1.3.1 Fast online signing from pairwise MtA. RompSig-Q aims at fast online signing and preprocesses multiplications. To achieve robustness through redundancy, following [65], the nonce k is generated as Shamir (instead of *additive*) secret shares. We denote by $[\![\cdot]\!]$ any secret state in the form of (t, n) Shamir shares held by the parties.[1]

In the key generation phase, each party publishes a CL encryption key, and they jointly run a Publicly Verifiable Secret Sharing (PVSS)-based DKG for the ECDSA threshold public key $X = g^x$, yielding shares of secret key $[\![x]\!]$. Then, at a high level, threshold signing works as follows:

Distributed randomness generation Sample random (t, n) Shamir shares $[\![k]\!]$ and $[\![\phi]\!]$ with joint PVSS.

[1] $[\![x]\!] := (x_1, x_2, \cdots, x_n)$; each party P_i holds a Shamir share x_i.

Share conversion and R reveal Open $R = g^k$, and obtain pairwise additive *fragments* of partial products in $[\![k]\!] \otimes [\![\phi]\!]$ and $[\![x]\!] \otimes [\![\phi]\!]$ with the MtA conversion.[2]

Online signing When m is ready, linearly transform the fragments to shares of $m\phi + rx\phi$ and $k\phi$ and publish them. Finally, locally reconstruct and divide these two values to get $s = k^{-1}(m + rx) \bmod q$.

The first two steps are message-independent presigning. We give some further intuition regarding this outline and how this translates to a three-round design.

Taking full advantage of PVSS. In a joint run of PVSS, each party publishes a PVSS dealing of a random value. In general, a PVSS dealing contains ciphertexts of Shamir shares encrypted under each of all parties's public keys and a NIZK proof that the plaintexts are indeed (t, n) Shamir shares. The aggregated result from adding up t dealings is guaranteed to be uniformly random as long as any party's contribution is honestly sampled as uniformly random.

The PVSS construction of Kate et al. [49] proves (t, n) Shamir shares are CL-encrypted by linking the plaintexts to a Feldman polynomial commitment. When aggregated, this polynomial commitment reveals each party's share mapped to an elliptic curve point via exponentiation. While one may consider using it to derive R or X in one round, this is not secure due to the *key-bias attack* [43, 50, 46, 3]. We mitigate this following the technique of Neji et al. [58]: We use another independent elliptic curve group generator h for the PVSS round, and let each party publish g raised to their share in the next round with a NIZK proof of discrete logarithm equality. The public values X and R can be derived via Lagrange interpolation from t such openings. This is robust, and uniform nonce distribution holds under the Decisional Diffie–Hellman assumption.

During presigning, we use the aggregated PVSS ciphertexts as the basis for the MtA conversion. It turns out that the Feldman polynomial commitment base h can be used to derive the MACs that will also go through the conversion and help identify faults in the online signing phase.[3]

Multiplicative-to-Additive share conversion. The MtA conversion is part of two-party private multiplication between each *ordered* pair of parties (P_i, P_j), where $i \neq j$. Suppose k_i is the private input of P_i, and ϕ_j is the input of P_j. As c_{ϕ_j}, the CL ciphertext of the share ϕ_j under P_j's public key, is publicly known, P_i can apply a scalar multiplication of k_i on it, and then apply a homomorphic subtraction of a randomly sampled $\mathbb{Z}_q$ element $\beta_{j,i}$ on the result. P_i then sends the ciphertext back to P_j. Now, $\beta_{j,i}$ is the additive share owned by P_i, and P_j can decrypt $\alpha_{j,i}$ from the ciphertext received; it holds that $k_i \cdot \phi_j = \alpha_{j,i} + \beta_{j,i}$.

Zero-knowledge proof for MtA. To ensure honest MtA and the validity of MACs that go through it, we devise a NIZK proof based on the work of Deng et al. [29]. Using the batching technique of Gennaro et al. [44], each party can prove a linear number of MtA conversions in near constant size. We further use an alternative formulation of the Fiat–Shamir heuristic to eliminate all group elements from the NIZK proofs and minimize their sizes.

Robust reconstruction of product. The technique is from [65]. We show how the product $k\phi$ can be reconstructed after the MtA conversion. Let $\mathcal{P}$ be the set of at least t qualified parties, and let $L_{i,\mathcal{P}}$ be the Lagrange coefficient of P_i with respect to $\mathcal{P}$. We have

$$\begin{aligned} k\phi &= (\textstyle\sum_{i\in\mathcal{P}} L_{i,\mathcal{P}} \cdot k_i) \cdot (\sum_{j\in\mathcal{P}} L_{j,\mathcal{P}} \cdot \phi_j) \\ &= \textstyle\sum_{i,j\in\mathcal{P}} L_{i,\mathcal{P}} \cdot L_{j,\mathcal{P}} \cdot k_i\phi_j. \end{aligned}$$

For each ordered pair (i, j) where $i \neq j$, the partial product $k_i\phi_j$ is the sum of the two pairwise-additive fragments P_i and P_j have obtained in the MtA conversion. Then the product can be assembled by combining the partial products multiplied by corresponding Lagrange coefficients. Since both multiplying a shared secret with a publicly-known constant and adding up two can be done locally, the same applies to the reconstruction of $m\phi + rx\phi$ after local linear operations.

Using the MACs, malicious shares that are inconsistent with the MtA conversion can be identified and discarded. As we assume at least t parties will give correct shares, after the filtering the reconstruction process will always succeed.

1.3.2 *Signing with constant-broadcast communication from threshold CL.*

Like Wong et al. [64], RompSig-L mainly builds on threshold CL encryption explored by Braun et al. [14]. CL encryption is "threshold-friendly" because of its ElGamal-like structure (see Figure 2). With a DKG over the CL group, the parties have shares of the secret key, and they can later collaborate to threshold-decrypt a ciphertext (c_1, c_2) under the threshold public key pk. Roughly, each party gives a partial decryption share like $c_1^{\mathsf{sk}_i}$, where sk_i is a share of the secret key sk, and they can be aggregated into c_1^{sk}.

We outline RompSig-L. In the key generation phase, the parties first run the threshold CL DKG, then the threshold ECDSA & ElGamal DKGs, generating a verification key $X = g^x$ and an ElGamal encryption key $Y = g^y$, with x and y secret-shared. They then collaborate to produce a CL ciphertext that encrypts the ECDSA signing key x under the threshold CL public key, i.e., $c_x = \mathsf{Enc}_{\mathsf{CL}}(x)$. Here, let $\mathsf{Enc}_{\mathsf{CL}}(a)$ denote a CL encryption of $a \in \mathbb{Z}_q$, and $\mathsf{Enc}_{\mathsf{EG}}(B)$ an ElGamal encryption of $B \in \mathbb{G}$, each under the respective threshold public key.

For each signing session, each party P_i does the following:

Nonce proposal Encapsulate a random nonce k_i in a CL ciphertext $\mathsf{Enc}_{\mathsf{CL}}(k_i)$, and g^{k_i} in an ElGamal ciphertext $\mathsf{Enc}_{\mathsf{EG}}(g^{k_i})$, prove the consistency, and broadcast them.

Nonce masking and R reveal Derive $c_k = \mathsf{Enc}_{\mathsf{CL}}(k)$ and $c_R = \mathsf{Enc}_{\mathsf{EG}}(R = g^k)$ by aggregating honest contributions attested by NIZK proofs. Give partial decryptions for c_R, so that all parties get R and r. Meanwhile, sample a mask ϕ_i, and publish $\mathsf{Enc}_{\mathsf{CL}}(\phi_i)$, $\mathsf{Enc}_{\mathsf{CL}}(k\phi_i)$ and $\mathsf{Enc}_{\mathsf{CL}}(x\phi_i)$, the latter two obtained via scaling c_k and c_x. Also prove honesty in NIZK.

Online signing Derive $c_\phi = \mathsf{Enc}_{\mathsf{CL}}(\phi)$, $c_u = \mathsf{Enc}_{\mathsf{CL}}(k\phi)$, and $c_v = \mathsf{Enc}_{\mathsf{CL}}(x\phi)$ via aggregation. When $m = \mathsf{H}(\mathtt{msg})$ is ready, linearly assemble a ciphertext c_w from c_ϕ and c_v that encapsulates $w = m\phi + rx\phi$. Give partial decryptions for c_w and c_u, so that all parties get $s = w/u$.

[2] Let $\otimes$ denote the outer product here. After the MtA conversion, each partial product $k_i\phi_j$ $(i \neq j)$ is split into two fragments held by P_i and P_j.

[3] This makes [49] a favorable choice compared to another CL-based PVSS [17] which does not yield a Feldman-like polynomial commitment.

With contributions to R wrapped inside ElGamal ciphertexts and R later threshold-decrypted, a rushing adversary cannot bias its distribution. Also, each party-specific local secret is only used once; this means any party can join or disengage at any time without breaking the shared state. This is how robustness under dynamic participation is achieved.

1.4 Paper Organization

The rest of this paper is organized as follows. Section 2 describes relevant cryptographic primitives and building blocks. Sections 3 and 4 describe RompSig-Q and RompSig-L, respectively. Section 5 presents the implementation and evaluation results. Appendix A gives details on instantiating the zero-knowledge proofs. Appendix B presents the security analysis.

2 PRELIMINARIES

General Notation. We use the multiplicative notation for group operations, and hence "point multiplication" is written as exponentiation. Let $\leftarrow$ and $:=$ denote assignments and $\leftarrow\!\$$ denote sampling from a distribution. We use uppercase letters for group elements and lowercase ones for scalars, except for ciphertexts $c_{(\cdot)}$ and group generators f, g, h. We use $\{a_j\}_{j\in J}$ to denote the *index set* of all elements a_j where $j \in J$, and may omit J when there's no equivocacy. We use $[m..n]$ to denote the integer interval $\{m, \cdots, n\}$, where $m < n$. Specifically, let $[n]$ be a shorthand for $[1..n]$.

2.1 ECDSA

The Elliptic Curve Digital Signature Algorithm (ECDSA) consists (KeyGen, Sign, Verify), as shown in Figure 1. The public parameters $\mathsf{pp} := (\mathbb{G}, q, g, \mathsf{H})$ define an elliptic curve group $\mathbb{G}$ generated by g with prime order q. Scalar arithmetics are in $\mathbb{Z}_q$, the field of integers modulo q. The hash function H is SHA-2 and maps arbitrary input to $\mathbb{Z}_q$. While $(r, \pm s)$ can both be valid signatures, we only consider the one with a smaller s valid.

KeyGen(pp)	Sign(x, msg)	Verify(X, msg, σ)
$x \leftarrow\!\$\ \mathbb{Z}_q$	$m \leftarrow \mathsf{H}(\mathsf{msg})$	**parse** $(r, s) \leftarrow \sigma$
$X \leftarrow g^x$	$k \leftarrow\!\$\ \mathbb{Z}_q$	$m \leftarrow \mathsf{H}(\mathsf{msg})$
return (X, x)	$R \leftarrow g^k$	$R' \leftarrow (g^m X^r)^{s^{-1}}$
	$r \leftarrow R\|_{x\text{-axis}}$	**if** $R'\|_{x\text{-axis}} = r$
	$s \leftarrow k^{-1}(m + xr)$	**return** 1
	return (r, s)	**return** 0

Figure 1: ECDSA

2.2 Network and Security Model

We assume synchronous communication, i.e., all messages in each round are delivered within a known timeout δ. We assume all parties have access to a reliable broadcast channel and (only in the setup phase) peer-to-peer private channels. We assume all protocol messages are authenticated. We remark that this network model is in line with all prior works with IA or robustness [16, 65, 64].

We consider a probabilistic polynomial-time (PPT) adversary that can corrupt up to $t - 1$ parties. We assume static corruption, i.e., the adversary chooses the parties to be corrupted before any interaction starts. The adversary is malicious, i.e., can learn the internal states of the corrupted parties, drop out of, or arbitrarily deviate from the protocol at any phase. We consider a *rushing* adversary: the message of a corrupted party can be decided after the adversary has learned all other parties' messages in each round, and the order in which the messages are delivered can be under adversarial control.

2.3 Definition of Threshold ECDSA

A (t, n) threshold ECDSA scheme consists of protocols (TKeyGen, TSign). The distributed key generation protocol TKeyGen lets the parties sample a (t, n) secret sharing of a signing key x and makes the corresponding verification key X public. It is sometimes also acceptable to let a trusted dealer generate and distribute the key shares. The threshold signing protocol TSign takes as input the message to be signed and, if certain conditions are met, outputs a signature of the message that is valid under the verification key.

We define unforgeability following Gennaro and Goldfeder [40] and robustness following Wong et al. [65, 64].

Definition 1 (Unforgeability). *We say a (t, n) threshold ECDSA scheme is $(t-1)$-unforgeable if, for any malicious adversary that statically corrupts up to $t-1$ parties and has the views of corrupted parties in* TKeyGen *and arbitrarily initiated* TSign *sessions for messages which the adversary adaptively chooses, the probability that the adversary can produce a valid signature for a previously unsigned message is negligible in the security parameter λ.*

Definition 2 (Robustness). *We say a (t, n) threshold ECDSA scheme is t-robust if, even in the presence of any malicious adversary that statically corrupts up to $t-1$ parties, a* TSign *session always outputs a valid signature for the input message as long as at least t parties remain throughout it and properly follow the protocol.*

2.4 Castagnos–Laguillaumie Encryption

The Castagnos–Laguillaumie (CL) linearly homomorphic encryption scheme [21, 22] consists of algorithms (KeyGen, Enc, Dec), shown in Figure 2. This scheme works in the group $\mathfrak{G} \simeq \mathfrak{G}^q \times \mathfrak{F}$, where $\mathfrak{F}$ is an order-q cyclic subgroup with easy discrete logarithm, and $\mathfrak{G}^q$ is a cyclic subgroup of unknown order. The public parameters $\mathsf{pp}_{\mathsf{CL}} := (\tilde{s}, f, g_q, q, \Delta_K)$ are

- $\tilde{s}$: an upper bound on the order of $\mathfrak{G}^q$;
- f : a generator of $\mathfrak{F}$ such that there exists an efficient deterministic algorithm $\mathsf{SolveDLog}_f(\cdot)$ that outputs $m \in \mathbb{Z}_q$ on input f^m;
- g_q : a generator of the subgroup $\mathfrak{G}^q$;
- q : the order of $\mathfrak{F}$, which we set equal to the order of the elliptic curve group $\mathbb{G}$;
- Δ_K : the discriminant of the class group.

We use $\mathcal{D}_q$ to denote the uniform distribution over $[2^{40} \cdot \tilde{s}]$, so that $\{g_q^r : r \leftarrow\!\$\ \mathcal{D}_q\}$ is statistically close to uniformly random in $\mathfrak{G}^q$. We may write $\mathsf{CL.Enc}(\mathsf{pk}, m; r)$ to give the encryption randomness a symbol. It is IND-CPA secure assuming Hard Subgroup Membership (HSM) in $\mathfrak{G}$ [22]. We define addition $\oplus$ and scalar multiplication

$\mathsf{KeyGen}(\mathsf{pp}_{\mathsf{CL}})$	$\mathsf{Enc}(\mathsf{pk}, m)$	$\mathsf{Dec}(\mathsf{sk}, c)$
$\mathsf{sk} \leftarrow\!\!\$\ \mathcal{D}_q$	$r \leftarrow\!\!\$\ \mathcal{D}_q$	**parse** $(c_1, c_2) \leftarrow c$
$\mathsf{pk} \leftarrow g_q^{\mathsf{sk}}$	$c \leftarrow (g_q^r, f^m \mathsf{pk}^r)$	$m \leftarrow \mathsf{SolveDLog}_f(c_2/c_1^{\mathsf{sk}})$
return $(\mathsf{pk}, \mathsf{sk})$	**return** c	**return** m

Figure 2: Castagnos–Laguillaumie Encryption.

$\otimes$ on CL ciphertexts: $(c_1, c_2) \oplus (d_1, d_2) := (c_1 d_1, c_2 d_2)$, and given $k \in \mathbb{Z}_q$ let $k \otimes (c_1, c_2) := (c_1^k, c_2^k)$. It is easy to check that the encryption preserves addition in the plaintext space $\mathbb{Z}_q$.

2.5 Zero-Knowledge Proofs

This work makes use of many zero-knowledge proofs, primarily Fiat–Shamir NIZKs based on interactive Σ-protocols. We give the relations as they are needed in the main body. The definition and construction details are in Appendix A.

2.6 Secret Sharing and Distributed Key Generation

Shamir's secret sharing. A (t, n) Shamir Secret Sharing [61] instance $\mathsf{SS}_{t,n}$ consists of algorithms $(\mathsf{Share}, \mathsf{Recon})$, where t is the reconstruction threshold and n is the total number of parties.

$\mathsf{Share}(s) \to \{s_j\}_{j\in[n]}$ **:** The dealer samples a random $(t-1)$-degree polynomial defined on $\mathbb{Z}_q$ with the constant term being the secret s, say $f(x) = s + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1}$, then computes a share $s_j = f(j)$ for each $j \in [n]$.

$\mathsf{Recon}(\{s_i\}_{i\in\mathcal{P}}) \to s$ **:** For each $j \in [n]$, we define the *Lagrange coefficient* $L_{j,\mathcal{P}} = \prod_{l\in\mathcal{P}, l\neq j} \frac{l}{l-j}$. If $|\mathcal{P}| < t$, return $\perp$ as fewer than t correct shares do not reveal any information about the secret s; otherwise, the original secret can be reconstructed as $s = \sum_{j\in\mathcal{P}} L_{j,\mathcal{P}} \cdot s_j$.

Feldman-VSS. Feldman's Verifiable Secret Sharing [36] provides security against malicious dealers or shareholders. Reconstruction with correct shares is the same.

$\mathsf{Share}(s) \to \left(\{s_j\}_{j\in[n]}, \{A_k\}_{k=0}^{t-1}\right)$ **:** The dealer first calls $\mathsf{SS}_{t,n}.\mathsf{Share}(s)$ to get $\{s_j\}_{j\in[n]}$. With the polynomial coefficients used, the dealer computes and publishes $\{A_k = g^{a_k}\}_{k=0}^{t-1}$, a.k.a. the polynomial commitment.

$\mathsf{VerifyShare}(\{A_k\}_{k=0}^{t-1}, j, s_j) \to \{0, 1\}$ **:** A share s_j can be verified by checking $g^{s_j} \stackrel{?}{=} \prod_{k=0}^{t-1} A_k^{j^k} = g^{f(j)}$.

PVSS based on CL encryption. Publicly-Verifiable Secret Sharing (PVSS) eliminates private channels in VSS and instead broadcasts ciphertexts of shares with a NIZK proof that the plaintexts are indeed (t, n) Shamir shares. We use the construction of Kate et al. [49] based on multi-receiver (randomness-reused) CL encryption. A joint run by at least t parties using random secrets gives a biasable DKG in one round (Figure 3). It takes the following input:

- g : the generator of the elliptic curve group $\mathbb{G}$;
- t : the target reconstruction threshold;
- sk_i : the CL secret key of the party P_i;
- $\{\mathsf{pk}_j\}_j$: CL public keys of parties who will hold shares.

$\mathsf{JointPVSS}(g, t, \mathsf{sk}_i, \{\mathsf{pk}_j\}_j) \to (X, x_i, \{X_j\}_j, \{c_{x_j}\}_j)$

..... Giving PVSS dealing .....

$\{a_{i,k}\}_{k=0}^{t-1} \leftarrow\!\!\$\ \mathbb{Z}_q^t$; $\{A_{i,k}\}_k \leftarrow \{g^{a_{i,k}}\}_k$;

$f_i(x) := \sum_{k=0}^{t-1} a_{i,k} \cdot x^k$; $\{s_{i,j}\}_j \leftarrow \{f_i(j)\}_{j\in[n]}$;

$r_i \leftarrow\!\!\$\ \mathcal{D}_q$; $R_i \leftarrow g_q^{r_i}$; $\{E_{i,j}\}_j \leftarrow \{f^{s_{i,j}} \mathsf{pk}_j^{r_i}\}_j$;

$\pi_i \leftarrow \mathsf{NIZK}_{\mathsf{PVSS}}.\mathsf{Prove}(\{\mathsf{pk}_j\}_j, R_i, \{E_{i,j}\}_j, g, \{A_{i,k}\}_k;\ \{s_{i,j}\}_j, r_i)$;

broadcast $\mathsf{dealing}_i = (R_i, \{E_{i,j}\}_j, \{A_{i,k}\}_k, \pi_i)$;

upon receiving $\mathsf{dealing}_j$: **check** π_j;

..... Aggregating for output .....

Let $\mathcal{P}$ be the set of t lowest-indexed qualified parties;

$\{c_{x_j}\}_j \leftarrow \{\bigoplus_{l\in\mathcal{P}} (R_l, E_{l,j})\}_{j\in\mathcal{P}}$; $x_i \leftarrow \mathsf{CL.Dec}(\mathsf{sk}_i, c_{x_i})$;

$X \leftarrow \prod_{j\in\mathcal{P}} A_{j,0}$; $\{X_j\}_j \leftarrow \{\prod_{k=0}^{t-1} (\prod_{l\in\mathcal{P}} A_{l,k})^{j^k}\}_{j\in\mathcal{P}}$;

return $(X, x_i, \{X_j\}_j, \{c_{x_j}\}_j)$

Figure 3: Joint PVSS subprotocol. Code for each P_i.

At its output, each party P_i gets a Shamir share x_i, along with public values $\{X_j = g^{x_j}\}_j$, $X = g^x$ and ciphertexts of shares $\{c_{x_j}\}_j$ under the shareholders' CL public keys; x is secret-shared and does not really appear.

A $\mathsf{NIZK}_{\mathsf{PVSS}}$ proof ensures that the dealing is honestly composed: Shamir shares consistent with the Feldman-like polynomial commitment are CL-encrypted under corresponding public keys, reusing the randomness. The relation is

$$\mathcal{R}_{\mathsf{PVSS}} = \{(\{\mathsf{pk}_j\}_j, R, \{E_j\}_j, g, \{A_k\}_k; \{s_j\}_j, r) : R = g_q^r \wedge \forall j\ (g^{s_j} = \textstyle\prod_k (A_k)^{j^k} \wedge E_j = f^{s_j} \mathsf{pk}_j^r)\}.$$

3 ROBUST THRESHOLD ECDSA FROM PAIRWISE MTA

In this section, we describe our first construction, RompSig-Q, and sketch the security proof.

3.1 RompSig-Q Description

We first overview how the protocols are supposed to work together. All n signatories, denoted $P_1, \cdots, P_n$ and given the public parameters, first run TKeyGen once, which includes publishing each party's CL public key and executing the DKG for threshold ECDSA itself. Afterwards, any quorum of at least t parties can perform presigning in batches and produce presignatures that contain $r = R|_{x\text{-axis}}$ and fragments of products for online signing.

Upon receiving a signing request with the message msg, a subset with at least t parties that have been in the same presigning quorum consume a fresh presignature, perform online signing, and output the full signature $\sigma = (r, s)$.

Periodically, all n signatories run KeyRefresh to generate new CL key pairs and ECDSA secret key shares, with the ECDSA verification key representing the entirety of all n parties remaining the same, in order to invalidate the shares that the adversary has obtained from corrupted parties.

$\mathsf{TKeyGen}(\mathsf{pp}) \to (\mathsf{sk}_i, \{\mathsf{pk}_j\}_j, X, x_i, \{X_j\}_j)$

.....Key generation for CL encryption .....

$(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow\!\$\ \mathsf{CL.KeyGen}(\mathsf{pp}_{\mathsf{CL}})$;

$\pi_{\mathsf{sk}_i} \leftarrow \mathsf{NIZK}_{\mathsf{Key}}.\mathsf{Prove}(g_q, \mathsf{pk}_i; \mathsf{sk}_i)$;

broadcast $(\mathsf{pk}_i, \pi_{\mathsf{sk}_i})$;

.....Joint PVSS.....

$(h^x, x_i, \{h^{x_j}\}_j, \{c_{x_j}\}_j) \leftarrow \mathsf{JointPVSS}(h, t, \mathsf{sk}_i, \{\mathsf{pk}_j\}_j)$;

..... Extracting ECDSA public key .....

$X_i \leftarrow g^{x_i}$; $\pi_{x_i} \leftarrow \mathsf{NIZK}_{\mathsf{DLEQ}}.\mathsf{Prove}(h, h^{x_i}, g, X_i; x_i)$;

broadcast $\mathsf{open}_i = (X_i, \pi_{x_i})$;

upon receiving open_j : **check** π_{x_j};

.....ECDSA public key output.....

Let $\mathcal{P}$ be the set of qualified parties; $X \leftarrow \prod_{j\in\mathcal{P}} X_j^{L_{j,\mathcal{P}}}$

Figure 4: RompSig-Q key generation. Code for each P_i.

Distributed Key Generation. The TKeyGen protocol requires (t, n), $\mathsf{pp}_{\mathsf{CL}}$, $(\mathbb{G}, q, g)$, and another generator $h \in \mathbb{G}$ that is independent from g, as public parameters. The elliptic curve group description is standard, the setup of CL public parameters is transparent, and h can be sampled from multi-party coin flipping or *hashing to curve*.

As shown in Figure 4, each party first samples a CL key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ and broadcasts pk_i with a proof of knowledge of the secret key. The relation is

$$\mathcal{R}_{\mathsf{Key}} = \{(g_q, \mathsf{pk}_i; \mathsf{sk}_i) : \mathsf{pk}_i = g_q^{\mathsf{sk}_i}\}.$$

After that, the parties invoke the PVSS-based DKG with the key-bias attack mitigation to generate the ECDSA verification key X and their respective shares.

Presigning Round 1 – Distributed randomness generation. Figure 5 shows the threshold signing protocol TSign, including the presign and online phases. To begin a signing session, the parties first use JointPVSS described in Figure 3 to sample random Shamir shares $[\![k]\!]$ and $[\![\phi]\!]$. Namely, each party composes two random PVSS dealings and broadcasts them in a combined message. Since we shall mitigate the key-bias attack by deferring the opening of R to the next round, the elliptic curve group generator h is used here.

Presigning Round 2 – Share conversion and R reveal. In the second round, each party's share of R (a Shamir share *in the exponent*) is published and the MtA is done. Each ordered pair of parties (P_i, P_j) securely multiplies their shares of nonces and gets fragments that can be used to open the product. In more detail, we have $k_i \cdot \phi_j = \beta_{j,i} + \alpha_{j,i}$ where P_i holds $\beta_{j,i}$ and P_j gets $\alpha_{j,i}$, and similarly $x_i \cdot \phi_j = \nu_{j,i} + \mu_{j,i}$ where P_i holds $\nu_{j,i}$ and P_j gets $\mu_{j,i}$. This is concretely instantiated by scaling the ciphertext of ϕ_j under pk_j known in Round 1 and then subtracting a random $\mathbb{Z}_q$ element which is the initiator's fragment. Then, the receiver's share can be obtained via decryption. We denote the class group identity by 1 here.

The same affine operations are also done on $\{\Phi_j\}_j$ in $\mathbb{G}$, and the corresponding results will be used for the consistency check in the online signing round. The initiator's fragments and $\{\Phi_j\}_j$ are stored to be used in the online phase.

Each party broadcasts R_i, the affinely operated multi-receiver ciphertexts, and the corresponding MACs, along with NIZK proofs for discrete logarithm equality base g and h, and for consistent affine operation on the multi-receiver ciphertext and the MACs with the correct scalar:

$$\mathcal{R}_{\mathsf{DLEQ}} = \{(h, K_i, g, R_i; k_i) : K_i = h^{k_i} \wedge R_i = g^{k_i}\};$$
$$\mathcal{R}_{\mathsf{Aff}} = \{(\{c_j, \tilde{c_j}, M_j, \tilde{M_j}\}_j, h, g, A; a, \{b_j\}_j) :$$
$$A = g^a \wedge \forall j\ (\tilde{c_j} = a \otimes c_j \oplus (1, f^{b_j}) \wedge \tilde{M_j} = M_j^a h^{b_j})\}.$$

Upon receiving $\mathsf{presignfin}_j$ from P_j, each party P_i checks its NIZK proofs, and if any proof fails verification, P_j is disqualified from $\mathcal{P}$ by all (honest) parties. Otherwise, P_i can decrypt and store fragments $\alpha_{i,j}$ and $\mu_{i,j}$ from the ciphertexts.

Presigning local finalization. After the presigning interactions terminate, the parties obtain R via Lagrange interpolation and then its x-coordinate r, and store them. Each party P_i also precomputes part of online-signing materials: P_i uses random (t, n) Shamir shares of 0 to *mask* local fragments of $u = k\phi$ that it holds,[4] and precomputes elliptic curve points $U_i = \Phi^{k_i}$ and $V_i = \Phi^{x_i}$, with NIZK proofs of their well-formedness, that will be used for the consistency check.

Online signing. In the online signing phase (Round 3 in Figure 5), each P_i computes the hash m of the message msg to be signed, and then uses random (t, n) Shamir shares $\{m_{i,j}\}_j$ of it to mask the fragments of $w = m\phi + rx\phi$ it holds. Each P_i broadcasts fragments $\{u_{i,j}\}_j, \{w_{i,j}\}_j$, the precomputed U_i, V_i and the related proofs.

Upon receiving the broadcast message $\mathsf{onlinesign}_j$ from P_j, each party first checks that U_j and V_j are well-formed, then uses the MACs to do a consistency check on P_j's fragments. To shed light on the consistency check, observe that honestly computed fragments $\{u_{i,l}\}_l$ held by P_i should satisfy

$$\begin{aligned}
&\textstyle\sum_l L_{l,\mathcal{P}} u_{i,l} + \sum_{l\neq i} L_{l,\mathcal{P}}(\alpha_{l,i} - \alpha_{i,l}) \\
&= \textstyle\sum_{l\neq i} L_{l,\mathcal{P}}(\alpha_{i,l} + \beta_{l,i} + \theta_{i,l}) + L_{i,\mathcal{P}}(k_i\phi_i + \theta_{i,i}) \\
&\quad + \textstyle\sum_{l\neq i} L_{l,\mathcal{P}}(\alpha_{l,i} - \alpha_{i,l}) \\
&= \textstyle\sum_{l\neq i} L_{l,\mathcal{P}}(\alpha_{l,i} + \beta_{l,i}) + L_{i,\mathcal{P}} k_i\phi_i \\
&= \textstyle\sum_{l\neq i} L_{l,\mathcal{P}} k_i\phi_l + L_{i,\mathcal{P}} k_i\phi_i \\
&= k_i\phi.
\end{aligned}$$

Likewise, fragments $\{w_{i,l}\}_l$ should satisfy

$$\begin{aligned}
&\textstyle\sum_l L_{l,\mathcal{P}} w_{i,l} + r\sum_{l\neq i} L_{l,\mathcal{P}}(\mu_{l,i} - \mu_{i,l}) \\
&= \textstyle\sum_{l\neq i} L_{l,\mathcal{P}}(m_{i,l}\phi_i + r(\nu_{l,i} + \mu_{i,l})) \\
&\quad + L_{i,\mathcal{P}}(m_{i,i}\phi_i + r\phi_i x_i) + r\sum_{l\neq i} L_{l,\mathcal{P}}(\mu_{l,i} - \mu_{i,l}) \\
&= m\phi_i + r\textstyle\sum_{l\neq i} L_{l,\mathcal{P}}(\nu_{l,i} + \mu_{l,i}) + rL_{i,\mathcal{P}}\phi_i x_i \\
&= m\phi_i + r\textstyle\sum_{l\neq i} L_{l,\mathcal{P}} x_i\phi_l + rL_{i,\mathcal{P}}\phi_i x_i \\
&= m\phi_i + r\phi x_i.
\end{aligned}$$

[4] To see why the masking is necessary, suppose the receiver P_j sees the fragments $u_{i,j} = \alpha_{i,j} + \beta_{j,i}$ and $u_{i,i} = k_i\phi_i$. With the knowledge of $\alpha_{j,i} + \beta_{i,j}$, P_j obtains $k_i\phi_j + k_j\phi_i$ and can actually solve k_i, ϕ_i, leading to a complete break.

$\mathsf{TSign}(\mathsf{sk}_i, \{\mathsf{pk}_j\}_j, X, x_i, \{X_j\}_j, \mathtt{msg}) \to \sigma$

..... Round 1: Distributed randomness generation .....

$(K, k_i, \{K_j\}_j, \{c_{k_j}\}_j) \leftarrow \mathsf{JointPVSS}(h, t, \mathsf{sk}_i, \{\mathsf{pk}_j\}_j)$;

$(\Phi, \phi_i, \{\Phi_j\}_j, \{c_{\phi_j}\}_j) \leftarrow \mathsf{JointPVSS}(h, t, \mathsf{sk}_i, \{\mathsf{pk}_j\}_j)$;

..... Round 2: Share conversion and R reveal .....

Let $\mathcal{P}$ be the set of Round-1 participants;

$R_i \leftarrow g^{k_i}$; $\pi_{R_i} \leftarrow \mathsf{NIZK}_{\mathsf{DLEQ}}.\mathsf{Prove}(h, K_i, g, R_i; k_i)$;

foreach $j \in \mathcal{P}\backslash\{i\}$:

$\beta_{j,i}, \nu_{j,i} \leftarrow\!\!\$\ \mathbb{Z}_q$;

$c_{\alpha_{j,i}} \leftarrow c_{\phi_j} \otimes k_i \oplus (1, f^{-\beta_{j,i}})$; $A_{j,i} \leftarrow \Phi_j^{k_i} h^{-\beta_{j,i}}$;

$c_{\mu_{j,i}} \leftarrow c_{\phi_j} \otimes x_i \oplus (1, f^{-\nu_{j,i}})$; $M_{j,i} \leftarrow \Phi_j^{x_i} h^{-\nu_{j,i}}$;

$\pi_{\alpha_{*,i}} \leftarrow \mathsf{NIZK}_{\mathsf{Aff}}.\mathsf{Prove}$

$\quad(\{c_{\phi_j}, c_{\alpha_{j,i}}, \Phi_j, A_{j,i}\}_j, h, h, K_i; k_i, \{-\beta_{j,i}\}_j)$;

$\pi_{\mu_{*,i}} \leftarrow \mathsf{NIZK}_{\mathsf{Aff}}.\mathsf{Prove}$

$\quad(\{c_{\phi_j}, c_{\mu_{j,i}}, \Phi_j, M_{j,i}\}_j, h, g, X_i; x_i, \{-\nu_{j,i}\}_j)$;

$\mathsf{presignfin}_i \leftarrow (R_i, \pi_{R_i}, \{c_{\alpha_{j,i}}, c_{\mu_{j,i}}, A_{j,i}, M_{j,i}\}_j, \pi_{\alpha_{*,i}}, \pi_{\mu_{*,i}})$;

broadcast $\mathsf{presignfin}_i$;

upon receiving $\mathsf{presignfin}_j$: **check** π_{R_j}, $\pi_{\alpha_{*,j}}$, $\pi_{\mu_{*,j}}$;

if any check fails, let $\mathcal{P} \leftarrow \mathcal{P}\backslash\{j\}$; otherwise:

$\alpha_{i,j} \leftarrow \mathsf{CL.Dec}(\mathsf{sk}_i, c_{\alpha_{i,j}})$; $\mu_{i,j} \leftarrow \mathsf{CL.Dec}(\mathsf{sk}_i, c_{\mu_{i,j}})$;

..... Presigning local finalization .....

$R \leftarrow \prod_{j\in\mathcal{P}} R_j^{L_{j,\mathcal{P}}}$; $r \leftarrow R|_{x\text{-axis}}$; $\{\theta_{i,j}\}_j \leftarrow \mathsf{SS}_{t,n}.\mathsf{Share}(0)$;

$\{u_{i,j}\}_j \leftarrow \{\alpha_{i,j} + \beta_{j,i} + \theta_{i,j}\}_{j\in\mathcal{P}\backslash\{i\}} \cup \{u_{i,i} = k_i\phi_i + \theta_{i,i}\}$;

$U_i \leftarrow \Phi^{k_i}$; $\pi_{U_i} \leftarrow \mathsf{NIZK}_{\mathsf{DLEQ}}.\mathsf{Prove}(g, R_i, \Phi, U_i; k_i)$;

$V_i \leftarrow \Phi^{x_i}$; $\pi_{V_i} \leftarrow \mathsf{NIZK}_{\mathsf{DLEQ}}.\mathsf{Prove}(g, X_i, \Phi, V_i; x_i)$;

..... Round 3: Online signing for $m = \mathsf{H}(\mathtt{msg})$ .....

$\{m_{i,j}\}_j \leftarrow \mathsf{SS}_{t,n}.\mathsf{Share}(m)$; $w_{i,i} = m_{i,i}\phi_i + r\phi_i x_i$;

$\{w_{i,j}\}_j \leftarrow \{m_{i,j}\phi_i + r(\nu_{j,i} + \mu_{i,j})\}_{j\in\mathcal{P}\backslash\{i\}} \cup \{w_{i,i}\}$;

broadcast $\mathsf{onlinesign}_i = (\{u_{i,j}\}_j, \{w_{i,j}\}_j, U_i, V_i, \pi_{U_i}, \pi_{V_i})$;

upon receiving $\mathsf{onlinesign}_j$: **check** π_{U_j}, π_{V_j},

$h^{\sum_{l\in\mathcal{P}} u_{j,l}\cdot L_{l,\mathcal{P}}} \prod_{l\in\mathcal{P}\backslash\{j\}} (A_{l,j}/A_{j,l})^{L_{l,\mathcal{P}}} \stackrel{?}{=} U_j$,

$h^{\sum_{l\in\mathcal{P}} w_{j,l}\cdot L_{l,\mathcal{P}}} \prod_{l\in\mathcal{P}\backslash\{j\}} (M_{l,j}/M_{j,l})^{r\cdot L_{l,\mathcal{P}}} \stackrel{?}{=} V_j^r \Phi_j^m$;

if any check fails, let $\mathcal{P} \leftarrow \mathcal{P}\backslash\{j\}$;

..... Finalizing signing .....

$w \leftarrow \sum_{j\in\mathcal{P}} (L_{j,\mathcal{P}} \cdot (\sum_{l\in\mathcal{P}} w_{j,l} \cdot L_{l,\mathcal{P}}))$;

$u \leftarrow \sum_{j\in\mathcal{P}} (L_{j,\mathcal{P}} \cdot (\sum_{l\in\mathcal{P}} u_{j,l} \cdot L_{l,\mathcal{P}}))$;

$s \leftarrow w \cdot u^{-1} \bmod q$; **return** $\sigma = (r, s)$

Figure 5: RompSig-Q threshold signing. Code for each P_i.

Raising h to the power of both sides for each of them, we have

$$h^{\sum_l L_{l,\mathcal{P}} u_{i,l}} \prod_{l\neq i} (A_{l,i}/A_{i,l})^{L_{l,\mathcal{P}}} \stackrel{?}{=} U_i;$$

$$h^{\sum_l L_{l,\mathcal{P}} w_{i,l}} \prod_{l\neq i} (M_{l,i}/M_{i,l})^{r\cdot L_{l,\mathcal{P}}} \stackrel{?}{=} V_i^r \Phi_i^m.$$

It can also be seen from the same reasoning that these Lagrange interpolated intermediate values can be interpolated once more to get $u = k\phi$ and $w = m\phi + r\phi x$. We hint that

$$\sum_i L_{i,\mathcal{P}} (\sum_{l\neq i} L_{l,\mathcal{P}} (\alpha_{l,i} - \alpha_{i,l})) = 0.$$

The consistency check determines the final set of qualified parties $\mathcal{P}$. The fragments associated with parties in $\mathcal{P}$ are used to open u and w, through two consecutive Lagrange interpolations.[5] The final output can thus be computed as (r, s), where $s = w/u$. If there are fewer than t qualified parties in $\mathcal{P}$, the session is aborted with the culprits flagged.

Similarly to other works, the online phase of this scheme has an *optimistic* shortcut that only involves < 1 ms of $\mathbb{Z}_q$ arithmetics when there are no faults. After receiving t online signing broadcasts, each party first tries to reconstruct the signature and verifies it with X. Since the presignature R and the message $\mathtt{msg}$ uniquely determine the correct signature, if the verification succeeds, the protocol can terminate at this point. Only when the signature verification fails is the aforementioned fallback mechanism triggered, and as long as t online participants remain qualified, the malformed shares can be properly excluded and the correct signature can be computed.

Proactive key refresh. The KeyRefresh subprotocol can be periodically run by at least t parties to render previously leaked secrets invalid, or to replenish lost shares, or to change the parameters t or n. It proceeds as follows. Each party P_i samples a new CL key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ and broadcasts pk_i with a proof of knowledge of sk_i. Using the refreshed CL key pairs, the parties run a variant of JointPVSS that preserves the threshold public key X: Each party P_i sets the free term of the VSS polynomial to the secret key share $\bar{x}_i$ it holds in the previous epoch, composes the new PVSS message using g as the group generator, and then broadcasts it. Malicious behavior can be identified by checking whether the constant term in the polynomial commitment is equal to $\bar{X}_i$, P_i's aggregated Shamir share in the exponent in the previous epoch, besides checking the $\mathsf{NIZK}_{\mathsf{PVSS}}$ proof. The new shares x_i and $\{X_j\}_j$ are finally aggregated from valid PVSS dealings via Lagrange interpolation instead of straightforward addition.

3.2 Security of RompSig-Q

We state the security of RompSig-Q in the theorem below.

Theorem 1. *RompSig-Q is a $(t-1)$-unforgeable, t-robust threshold ECDSA scheme, assuming the enhanced existential unforgeability of single-party ECDSA, Decisional Diffie–Hellman (DDH) in $\mathbb{G}$, and Hard Subgroup Membership (HSM) in $\mathfrak{G}$, in the random oracle model.*

We prove the unforgeability in subsection B.1, subsection B.2, and prove the robustness in subsection B.4. Our overall idea to prove unforgeability is in line with prior game-based security proofs [40, 19, 65]. As we will explain in subsection B.1, we build a reduction of the unforgeability of our threshold ECDSA scheme to the enhanced existential unforgeability of single-party ECDSA [16], which models the security of centralized ECDSA used with presignatures as a hardness assumption. Namely, if an adversary breaks the unforgeability of RompSig-Q with non-negligible probability, then using

[5] The results in this step can be different from something similar that has been computed for the consistency check, if malicious parties are detected.

it we can build a forger that breaks the aforementioned assumption, resulting in a contradiction. We construct the reduction via a simulation argument. This requires us to embed a verification key, obtained as a challenge with unknown discrete logarithm in the single-party security game, into a simulated TKeyGen session and, at each signature query made by the adversary, embed a presignature R and a signature s obtained from the single-party security game into a TSign session. In subsection B.2 we describe the simulator and argue that the simulation succeeds and is indistinguishable from a real execution, unless with negligible probability, under the DDH and HSM assumptions. The robustness property can be proven in a way similar to [65, 64].

4 ROBUST THRESHOLD ECDSA IN THE CDN PARADIGM

Threshold CL encryption. Braun et al. [14] first constructed threshold CL encryption and scalable MPC protocols based on it. The threshold CL encryption scheme can be initialized with a Pedersen-like DKG in the class group $\mathfrak{G}$. The main technicality is that $\mathfrak{G}$ is a *hidden-order group*: The group order is not efficiently computable even given all information used to generate it. For prime-order groups, computing Lagrange coefficients requires inversion in the scalar field (modulo a prime), while in $\mathfrak{G}$, Lagrange interpolation is not directly feasible. To deal with this challenge, [14] modifies Feldman-VSS by introducing a multiplicative factor $\Delta = n!$ to secrets. Multiplying Δ with Lagrange coefficients calculated in $\mathbb{Q}$ always gives results in $\mathbb{Z}$.

We build on a modified version of [14]. Their security proof assumes an honest majority, but [64, 13] extended it to the dishonest-majority case. Their original scheme uses interactive Σ-protocols, and following [64, 13] we use Fiat-Shamir NIZKs. It is not necessary to use a DKG for class groups that ensures uniform distribution like in [64] (since it is rekeyable [48]). On the other hand, our threshold CL DKG is not robust.

4.1 RompSig-L Description

Distributed Key Generation. As shown in Figure 6, the parties first generate the threshold CL public key using a variant of Pedersen-DKG that shares secrets in $\mathbb{Z}$.[6] Since each pair of public and secret shares satisfy $\mathsf{pk}_i = g_q^{\mathsf{sk}_i}$, we can reuse these key pairs for the PVSS-based DKG for ECDSA and ElGamal encryption. Besides that, the key generation protocol also outputs a CL ciphertext c_x of the ECDSA secret key x under the threshold CL public key pk, which will be used later in each signing session. The Lagrange coefficients used here are simply obtained from $\mathbb{Z}_q$ arithmetics.

Our threshold CL DKG in Figure 6 differs from the one of [14] in the use of the proof of knowledge $\mathsf{NIZK}_{\mathsf{Key}}$: Each party only proves the knowledge of the constant term first and of the secret key share later, unlike [14] which does the same for all coefficients of the polynomial. The relation is

$$\mathcal{R}_{\mathsf{Key}} = \{(g_q, C_{i,0}; \alpha_{i,0}) : C_{i,0} = g_q^{\alpha_{i,0}}\}.$$

[6] To enable security against a dishonest majority, we sacrifice robustness in the DKG phase for threshold CL encryption by allowing any complaint to abort it. Since the DKG of CL encryption is a one-time setup, we consider this acceptable.

$\mathsf{TKeyGen}(\mathsf{pp}) \to (\mathsf{sk}_i, \mathsf{pk}, \{\mathsf{pk}_j\}_j, X, x_i, Y, y_i, \{Y_j\}_j, c_x)$

.....DKG for threshold CL encryption .....

$\Delta := n!$; $\alpha_{i,0} \leftarrow\!\$\ \mathcal{D}_q$; $\{\alpha_{i,k}\}_{k=1}^{t-1} \leftarrow\!\$\ \mathcal{D}_{\mathsf{ss}}^{t-1}$;

$f_i(x) := \alpha_{i,0} \cdot \Delta + \sum_{k=1}^{t-1} \alpha_{i,k} \cdot x^k$; $\{s_{i,j}\}_j \leftarrow \{f_i(j)\}_{j\in[n]}$;

$\{C_{i,k}\}_k \leftarrow \{C_{i,0} = g_q^{\alpha_{i,0}}\} \cup \{g_q^{\Delta\cdot\alpha_{i,k}}\}_{k=1}^{t-1}$;

broadcast $\pi_{\alpha_{i,0}} = \mathsf{NIZK}_{\mathsf{Key}}.\mathsf{Prove}(g_q, C_{i,0}; \alpha_{i,0})$;

broadcast $\mathsf{polycmt}_i = \{C_{i,k}\}_k$; **send** $s_{i,j}$ to each P_j;

upon receiving $\mathsf{polycmt}_j$, $s_{j,i}$:

  check $g_q^{s_{j,i}\cdot\Delta} \stackrel{?}{=} C_{j,0}^{\Delta^2} \cdot \prod_{k=1}^{t-1} C_{j,k}^{i^k}$, abort on inconsistency;

.....Joint PVSS.....

$\mathsf{sk}_i \leftarrow \sum_{j\in[n]} s_{j,i} \cdot \Delta$; $\mathsf{pk} \leftarrow \prod_{j\in[n]} C_{j,0}$;

$\{\mathsf{pk}_j\}_j \leftarrow \{(\prod_{l\in[n]} C_{l,0})^{\Delta^2} \cdot \prod_{k=1}^{t-1} (\prod_{l\in[n]} C_{l,k})^{j^k}\}_{j\in[n]}$;

broadcast $\pi_{\mathsf{sk}_i} = \mathsf{NIZK}_{\mathsf{Key}}.\mathsf{Prove}(g_q, \mathsf{pk}_i; \mathsf{sk}_i)$;

$(h^x, x_i, \{h^{x_j}\}_j, \{c_{x_j}\}_j) \leftarrow \mathsf{JointPVSS}(h, t, \mathsf{sk}_i, \{\mathsf{pk}_j\}_j)$;

$(Y, y_i, \{Y_j\}_j, \{c_{y_j}\}_j) \leftarrow \mathsf{JointPVSS}(g, t, \mathsf{sk}_i, \{\mathsf{pk}_j\}_j)$;

.....Extracting EC public key.....

$X_i \leftarrow g^{x_i}$; $c_i \leftarrow \mathsf{CL.Enc}(\mathsf{pk}, x_i; r_i)$;

$\pi_{x_i} \leftarrow \mathsf{NIZK}_{\mathsf{CL\text{-}DLEQ}}.\mathsf{Prove}(g, X_i, h, h^{x_i}, \mathsf{pk}, c_i; x_i, r_i)$;

broadcast $\mathsf{open}_i = (X_i, c_i, \pi_{x_i})$;

upon receiving open_j : **check** π_{x_j};

.....EC public key output.....

Let $\mathcal{P}$ be the set of qualified parties;

$X \leftarrow \prod_{j\in\mathcal{P}} X_j^{L_{j,\mathcal{P}}}$; $c_x \leftarrow \bigoplus_{j\in\mathcal{P}} (c_j \otimes L_{j,\mathcal{P}})$

Figure 6: RompSig-L key generation. Code for each P_i.

The distribution $\mathcal{D}_{\mathsf{ss}}$ of coefficients $\{a_{i,k}\}_{k=1}^{t-1}$ is uniform over $[2^{42} \cdot t^2\Delta \cdot |\mathcal{D}_q|]$ to statistically hide the secret $\alpha_{i,0}$. We refer the reader to [13, Definition 3] for details.

Presigning Round 1 – Nonce proposal. The first round in Figure 7 generates uniformly random k and $R = g^k$, each encapsulated in a ciphertext under the threshold public key, from the parties' contributions. Each party P_i samples a random k_i, and creates a CL–ElGamal double encryption of it using the threshold CL & ElGamal public keys pk and Y. The ElGamal ciphertext is effectively one of $R_i = g^{k_i}$ and homomorphically preserves the group law of $\mathbb{G}$.

A NIZK proof is used to ensure that the same k_i is double-encrypted in the CL and ElGamal ciphertexts. Upon receiving other parties' messages, the NIZK proofs are checked to decide the set of qualified round-1 participants $\mathcal{P}$. The relation is

$$\mathcal{R}_{\mathsf{CL\text{-}EG}} = \{(\mathsf{pk}, c_{k_i}, Y, c_{R_i}; k_i, \rho_i, r_i) : c_{k_i} = (g_q^{\rho_i}, f^{k_i}\mathsf{pk}^{\rho_i}) \wedge c_{R_i} = (g^{r_i}, g^{k_i}Y^{r_i})\}.$$

This method of randomness generation is robust and resistant to adversarial biasing: Since contributions to R are encapsulated in ElGamal ciphertexts and then R is opened in a (t, n) threshold way, the rushing adversary can neither learn anything about R in the proposing phase (due to the IND-CPA security of both encryption

$\mathsf{TSign}(\mathsf{sk}_i, \mathsf{pk}, \{\mathsf{pk}_j\}_j, Y, y_i, \{Y_j\}_j, c_x, \mathtt{msg}) \to \sigma$

..... Round 1: Nonce proposal .....

$k_i, r_i \leftarrow\!\!\$\ \mathbb{Z}_q$; $c_{k_i} \leftarrow \mathsf{CL.Enc}(\mathsf{pk}, k_i; \rho_i)$; $c_{R_i} \leftarrow (g^{r_i}, g^{k_i} Y^{r_i})$;

$\pi_{k_i} \leftarrow \mathsf{NIZK}_{\text{CL-EG}}.\mathsf{Prove}(\mathsf{pk}, c_{k_i}, Y, c_{R_i}; k_i, \rho_i, r_i)$;

broadcast $\mathsf{prop}_i = (c_{k_i}, c_{R_i}, \pi_{k_i})$;

upon receiving prop_j : **check** π_{k_j};

..... Round 2: Nonce masking and R reveal .....

Let $\mathcal{P}$ be the set of qualified parties in Round 1;

$c_k \leftarrow \bigoplus_{j\in\mathcal{P}} c_{k_j}$; $c_R = (E, \Gamma) \leftarrow \bigoplus_{j\in\mathcal{P}} c_{R_j}$;

$\Omega_i \leftarrow E^{y_i}$; $\pi_{\Omega_i} \leftarrow \mathsf{NIZK}_{\mathsf{DLEQ}}.\mathsf{Prove}(g, Y_i, E, \Omega_i; y_i)$;

$\phi_i \leftarrow\!\!\$\ \mathbb{Z}_q$; $c_{\phi_i} \leftarrow \mathsf{CL.Enc}(\mathsf{pk}, \phi_i; \hat{\phi}_i)$; $\Phi_i \leftarrow g^{\phi_i}$;

$c_{u,i} \leftarrow \phi_i \otimes c_k$; $c_{v,i} \leftarrow \phi_i \otimes c_x$;

$\pi_{\phi_i} \leftarrow \mathsf{NIZK}_{\text{CL-Scal2}}.\mathsf{Prove}$

$\quad(\mathsf{pk}, c_{\phi_i}, g, \Phi_i, c_k, c_{u,i}, c_x, c_{v,i}; \phi_i, \hat{\phi}_i)$;

broadcast $\mathsf{presignfin}_i = (\Omega_i, c_{\phi_i}, c_{u,i}, c_{v,i}, \pi_{\Omega_i}, \Phi_i, \pi_{\phi_i})$;

upon receiving $\mathsf{presignfin}_j$: **check** $\pi_{\Omega_j}, \pi_{\phi_j}$;

..... Presigning local finalization .....

Let $\mathcal{P}'$ be the set of qualified parties in Round 2;

$R \leftarrow \Gamma / \prod_{j\in\mathcal{P}'} \Omega_j^{L_{j,\mathcal{P}'}}$; $r \leftarrow R|_{x\text{-axis}}$;

$c_u = (C_u, \Lambda_u) \leftarrow \bigoplus_{j\in\mathcal{P}'} c_{u,j}$; $U_i \leftarrow C_u^{\mathsf{sk}_i}$;

$c_\phi \leftarrow \bigoplus_{j\in\mathcal{P}'} c_{\phi,j}$; $c_v \leftarrow \bigoplus_{j\in\mathcal{P}'} c_{v,j}$;

..... Round 3: Online signing for $m = \mathsf{H}(\mathtt{msg})$ .....

$c_w = (C_w, \Lambda_w) \leftarrow m \otimes c_\phi \oplus r \otimes c_v$; $W_i \leftarrow C_w^{\mathsf{sk}_i}$;

$\pi_{d_i} \leftarrow \mathsf{NIZK}_{\mathsf{CGDL3}}.\mathsf{Prove}(g_q, \mathsf{pk}_i, C_u, U_i, C_w, W_i; \mathsf{sk}_i)$;

broadcast $\mathsf{onlinesign}_i = (U_i, W_i, \pi_{d_i})$;

upon receiving $\mathsf{onlinesign}_j$: **check** π_{d_j};

..... Finalizing signing .....

Let Q be the set of qualified parties in Round 3; $\Delta := n!$;

$\hat{w} \leftarrow \mathsf{SolveDLog}_f(\Lambda_w^{\Delta^3} / \prod_{j\in Q} W_j^{\Delta\cdot\mathcal{L}_{j,Q}})$;

$\hat{u} \leftarrow \mathsf{SolveDLog}_f(\Lambda_u^{\Delta^3} / \prod_{j\in Q} U_j^{\Delta\cdot\mathcal{L}_{j,Q}})$;

$s \leftarrow \hat{w} \cdot \hat{u}^{-1} \bmod q$; **return** $\sigma = (r, s)$

Figure 7: RompSig-L threshold signing. Code for each P_i.

schemes), nor adaptively choose to withdraw a submitted contribution, nor prevent enough honest parties from opening it *as is* via threshold decryption.

Presigning Round 2 – Nonce masking and R reveal. As the first round of interaction finishes, a CL ciphertext c_k of $k = \sum_{j\in\mathcal{P}} k_j$ under pk and an ElGamal ciphertext c_R of $R = g^k = \prod_{j\in\mathcal{P}} g^{k_j}$ under Y are aggregated from the contributions of qualified parties. Each party P_i then computes a *partial decryption share* Ω_i, a Shamir share of what can be used to decrypt c_R, with a proof that the correct secret key share y_i is used.

Each party P_i samples ϕ_i, a share of the multiplication mask ϕ. P_i then CL-encrypts it under pk, and also homomorphically scales ciphertexts c_k and c_x by ϕ_i so as to get ciphertexts $c_{u,i}$ and $c_{v,i}$ which encapsulate $k\phi_i$ and $x\phi_i$ respectively. P_i proves in NIZK that the same ϕ_i is used for these three ciphertexts; verifying the proofs defines the round-2 set of qualified parties $\mathcal{P}'$. The relation is

$$\mathcal{R}_{\text{CL-Scal2}} = \{(\mathsf{pk}, c_{\phi_i}, g, \Phi_i, c_k, c_{u,i}, c_x, c_{v,i}; \phi_i, \hat{\phi}_i) : c_{\phi_i} = (g_q^{\hat{\phi}_i}, f^{\phi_i}\mathsf{pk}^{\hat{\phi}_i}) \wedge \Phi_i = g^{\phi_i} \wedge c_{u,i} = \phi_i \otimes c_k \wedge c_{v,i} = \phi_i \otimes c_x\}.$$

Presigning local finalization. With at least t partial decryption shares, R is decrypted from c_R. CL ciphertexts c_ϕ, c_u and c_v are aggregated from valid contributions via homomorphic addition: c_u encapsulates $k\phi$ and c_v encapsulates $x\phi$. The partial decryption share for c_u and a scalar multiplication of c_v by r are precomputed ahead of online signing.

Online signing. When msg arrives, a hash m and the ciphertext c_w that encapsulates $m\phi + r\phi x$ can be computed, where $r = R|_{x\text{-axis}}$. Each party P_i gives partial decryption shares U_i and W_i for c_u and c_w respectively, with a proof that the sk_i used is consistent with the public key share pk_i. Verifying the proofs defines the online-phase set of qualified parties Q. The relation is

$$\mathcal{R}_{\mathsf{CGDL3}} = \{(g_q, \mathsf{pk}_i, C_u, U_i, C_w, W_i; \mathsf{sk}_i) : \mathsf{pk}_i = g_q^{\mathsf{sk}_i} \wedge U_i = C_u^{\mathsf{sk}_i} \wedge W_i = C_w^{\mathsf{sk}_i}\}.$$

We explain how the partial decryption shares indeed give rise to the correct output s. Observe in Figure 6 that

$$g_q^{\sum_{j\in Q} \mathsf{sk}_j \cdot \Delta \cdot \mathcal{L}_{j,Q}} = \mathsf{pk}^{\Delta^3}.$$

Hence, by lifting both parts of the ciphertext to the power of Δ^3 we can cancel out the randomness and obtain $f^{w\cdot\Delta^3}$ and $f^{u\cdot\Delta^3}$; solving the discrete logarithm and then dividing the two results cancels out Δ^3 and gives s, as intended. The Lagrange coefficients in this round follow arithmetics in $\mathbb{Q}$, and we use $\mathcal{L}_{j,Q}$ to denote them. The result of the factorial Δ multiplied by them are always in $\mathbb{Z}$.

Proactive key refresh. Similarly to RompSig-Q, this scheme also supports a proactive key refresh. The parties first run a new threshold CL DKG, then use the new CL key pairs to perform the same X-preserving variant of JointPVSS as in the key refresh of RompSig-L. A new threshold ElGamal encryption key is generated. Finally, each party P_i broadcasts a ciphertext c_i of its new share x_i under the new CL public key so that a new ciphertext c_x can be interpolated.

4.2 Security of RompSig-L

We state our security theorem for RompSig-L below.

Theorem 2. *RompSig-L is a $(t-1)$-unforgeable, t-robust threshold ECDSA scheme, assuming the enhanced existential unforgeability of single-party ECDSA, Decisional Diffie–Hellman (DDH) in $\mathbb{G}$, Hard Subgroup Membership (HSM) and Rough Order (RO) in $\mathfrak{G}$, in the random oracle model.*

We prove the unforgeability in subsection B.1, subsection B.3, and prove the robustness in subsection B.4. The high-level idea of our unforgeability proof here is in line with that of RompSig-Q, except that the simulation is a lot simpler due to the use of threshold homomorphic encryption. This is also observed in [64] as a "simple

Table 2: Our outbound broadcast communication in bytes.

Round	RompSig-Q	RompSig-L
#1	$64t + 440m + 872$	752
#2	$504m + 528$	1723
#3	$64m + 192$	624
Total	$64t + 1008m + 1592$	3099

security proof" property. Since our tools are the same as in [64], their proof of robustness also applies here.

5 IMPLEMENTATION AND EVALUATION

This section gives theoretical and experimental results that demonstrate the efficiency of our schemes. We implement our schemes with ~2,300 lines of Rust, excluding dependencies such as bindings to the optimized C++ implementation of the CL cryptosystem [12]. We focus on 128-bit security and the elliptic curve `Secp256k1`.

5.1 Bandwidth Cost

We analyze the communication cost of our schemes. Each elliptic curve point takes 33 bytes with point compression enabled. Each class group element serializes to 1,755 bits (220 bytes) using the compression method of Dobson et al. [32]. Several big integers in the NIZK proofs have sizes of up to 152 bytes.

We sum up the sizes of all components of each message and obtain each party's total outbound communication in Table 2, where m is the number of session participants and $t \le m \le n$. Taking $m = t$ to match non-robust schemes, the total broadcast communication of each party is $(1.0t + 1.6)$ KiB for RompSig-Q and 3.0 KiB for RompSig-L. If the protocol messages are relayed by centralized servers, this is indeed what each party has to communicate. On the other hand, in peer-to-peer networks, broadcasts must be realized using Byzantine Reliable Broadcast protocols; each broadcast typically takes $O(\kappa n^2 + n|M|)$ communication, where κ is the output length of hash functions and $|M|$ is the length of each message [28]. Identifiable abort and robustness require broadcasts in the dishonest-majority case [26]. If these features are given up in favor of less overhead, echo broadcasts can be used instead; see the full version of [16].

We do not compare in terms of bandwidth usage with schemes that do not use broadcast channels [56, 35]. DKLs24 [35] takes $50 \cdot (t - 1)$ KiB of peer-to-peer communication using OT extension, but if other OLE instantiations are used, it can consume less bandwidth than ours since it does not come with broadcast overhead; however, without broadcasts, it cannot achieve identifiable abort or robustness.

5.2 Computational Cost

We measure each party's single-threaded computation time for quorums of sizes 3 to 20, averaged over 100 locally simulated runs on a test platform running Ubuntu 24.04 with an i7-13700K CPU and 32 GB of RAM.

5.2.1 Benchmarking RompSig-Q. We compare it with the CGGMP20 protocol of Canetti et al. [16]. This scheme is used in production,

Table 3: Comparison of broadcast communication in KiB.

Scheme	3-party	5-party	7-party
CGGMP20 [16]	44.2	86.7	129.2
WMYC23 [65]	14.7	16.9	19.1
WMC24 [64]	3.4	3.4	3.4
RompSig-Q	4.6	6.6	8.6
RompSig-L	3.0	3.0	3.0

At 128-bit security (3072-bit Paillier modulus and 1827-bit class-group fundamental discriminant). Numbers of CGGMP20 are based on [19].

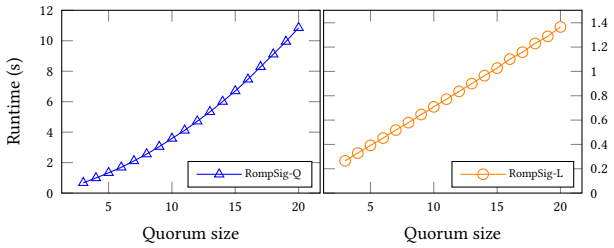

Figure 8: Running time measurements of our schemes.

has a similar round count of four, and also uses NIZKs for accountability. We use the audited open-source implementation of [16] from Dfns[7], and set $m = t$. Since the online phases of both schemes take less than 1 ms computation and are practically negligible, the running time is mostly attributed to the presigning phase.

Our first scheme RompSig-Q outperform the 4-round protocol of CGGMP20 [16] by 20% to 30% for small deployments (3 to 7 parties) which are common for cryptocurrency wallets (Table 4). Our advantage is mainly due to not having to deal with range proofs and that CL is faster than Paillier. The edge diminishes when more parties are involved, as CGGMP20 only invokes the $O(n^2)$ cheater identification process when there is a failure, but ours need to verify all NIZKs in each round to ensure it proceeds in a robust way.

The online signing phases of both schemes are very fast (< 1 ms) when there are no faults, since only $\mathbb{Z}_q$ arithmetics are involved. On the other hand, our scheme is robust when run with redundant parties while [16] can only abort on misbehavior. The fallback checking process triggered by failure of optimistic online signing in RompSig-Q is reasonably fast as well, taking $(1.4, 11.1, 47.6)$ ms single-threaded computation with $(3, 10, 20)$ parties, respectively.

5.2.2 Benchmarking RompSig-L. Our second scheme RompSig-L requires non-trivial computation in the online signing phase. While some of the computation can be implicitly preprocessed, the online signing latency is still on the order of 20 ms on the happy path, and if malicious nodes prevent success on first trial, it can be even higher. On the other hand, the total computation per party scales linearly in RompSig-L, and the total running time is significantly lower than RompSig-Q for all quorum sizes.

We also compare RompSig-L with our implementation of the WMC24 scheme [64]. Their single-threaded running time per party

[7] https://github.com/LFDT-Lockness/cggmp21.

Table 4: Comparing single-threaded signing runtime in ms.

Scheme	3-party	5-party	7-party
CGGMP20 [16] (4-round)	918	1810	2720
CCLST20 [18]	388	628	868
DKLs24 [35]	21	41	61
WMC24 [64]	281	411	535
RompSig-Q	678	1334	2110
RompSig-L	265	391	515

At 128-bit security. All CL-based schemes are implemented with the new library [12]. Based on local runs, and as such do not take network latency into account.

is (281, 411, 535) ms for (3, 5, 7) parties, while ours is (265, 391, 515) ms. Our computational cost is approximately 5% lower, and in addition, our advantage of one fewer round of interaction will make it stand out in real-world deployments.

As a test of scalability, we run a simulation of 128-party signing with all the 24 threads of our i7-13700K CPU. The multi-threaded computation cost of each party is 1.02 seconds, demonstrating that large-scale deployment is feasible. We remark that [35] scales even better computationally, but would require 6 megabytes of peer-to-peer communication, while ours only requries 3 KiB of broadcast communication. In a somewhat centralized scenario wherein broadcasts are handled by servers, ours will be more competitive.

5.2.3 Potential Optimizations. In our protocols, the majority of computation is due to the need to continually verify each other's NIZK proofs of honesty, in order to determine the possibly changing set of qualified parties. In somewhat centralized deployments, if the servers can be trusted for the detection of malicious misbehavior, then the clients may not have to waste CPU time on NIZK verification and have much better performance. This, of course, is a special assumption and is at odds with the common model, but may be of interest in some scenarios. In fact, we believe this accounts for many real-world deployments.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers of ACM CCS 2024, NDSS 2025, Usenix Security 2025, and ACM CCS 2025 for their constructive comments, and Bitmain and Antalpha for supporting this work.

REFERENCES

[1] Damiano Abram, Ariel Nof, Claudio Orlandi, Peter Scholl, and Omer Shlomovits. 2022. Low-bandwidth threshold ECDSA via pseudorandom correlation generators. In *IEEE S&P 2022*, 2554–2572. DOI: 10.1109/SP46214.2022.9833559.

[2] Michael Adjedj, Constantin Blokh, Geoffroy Couteau, Antoine Joux, and Nikolaos Makriyannis. 2024. Two-round 2PC ECDSA at the cost of 1 OLE. Cryptology ePrint Archive, Paper 2024/1950. (2024).

[3] Diego F. Aranha, Felipe Rodrigues Novaes, Akira Takahashi, Mehdi Tibouchi, and Yuval Yarom. 2020. LadderLeak: breaking ECDSA with less than one bit of nonce leakage. In *ACM CCS 2020*. ACM, 225–242. DOI: 10.1145/3372297.3417268.

[4] Jean-Philippe Aumasson, Adrian Hamelink, and Omer Shlomovits. 2020. A survey of ECDSA threshold signing. Cryptology ePrint Archive, Paper 2020/1390. (2020).

[5] Judit Bar-Ilan and Donald Beaver. 1989. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *ACM PODC '89*. ACM, 201–209. DOI: 10.1145/72981.72995.

[6] Mihir Bellare, Alexandra Boldyreva, Kaoru Kurosawa, and Jessica Staddon. 2007. Multirecipient encryption schemes: how to save on bandwidth and computation without sacrificing security. *IEEE Transactions on Information Theory*, 53, 11, 3927–3943. DOI: 10.1109/TIT.2007.907471.

[7] Constantin Blokh, Nikolaos Makriyannis, and Udi Peled. 2022. Efficient asymmetric threshold ECDSA for MPC-based cold storage. Cryptology ePrint Archive, Paper 2022/1296. (2022).

[8] Alexandra Boldyreva. 2002. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *PKC 2003*. Springer, 31–46. DOI: 10.1007/3-540-36288-6_3.

[9] Dan Boneh, Rosario Gennaro, and Steven Goldfeder. 2017. Using level-1 homomorphic encryption to improve threshold DSA signatures for bitcoin wallet security. In *LATINCRYPT 2017* (LNCS 11368). Springer, 352–377. DOI: 10.1007/978-3-030-25283-0_19.

[10] Dan Boneh, Iftach Haitner, and Yehuda Lindell. 2024. Exponent-VRFs and their applications. Cryptology ePrint Archive, Paper 2024/397. (2024).

[11] Alexandre Bouez and Kalpana Singh. 2023. One round threshold ECDSA without roll call. In *CT-RSA 2023* (LNCS 13871). Springer, 389–414. DOI: 10.1007/978-3-031-30872-7_15.

[12] Cyril Bouvier, Guilhem Castagnos, Laurent Imbert, and Fabien Laguillaumie. 2023. I want to ride my BICYCL: BICYCL implements cryptography in class groups. *Journal of Cryptology*, 36, 3, 1–31. DOI: 10.1007/s00145-023-09459-1.

[13] Lennart Braun, Guilhem Castagnos, Ivan Damgård, Fabien Laguillaumie, Kelsey Melissaris, Claudio Orlandi, and Ida Tucker. 2024. An improved threshold homomorphic cryptosystem based on class groups. In *SCN 2024* (LNCS 14974). Springer, 24–46. DOI: 10.1007/978-3-031-71073-5_2.

[14] Lennart Braun, Ivan Damgård, and Claudio Orlandi. 2023. Secure multiparty computation from threshold encryption based on class groups. In *CRYPTO 2023* (LNCS 14081). Springer, 613–645. DOI: 10.1007/978-3-031-38557-5_20.

[15] Jan Camenisch and Markus Stadler. 1997. Proof systems for general statements about discrete logarithms. *Technical Report/ETH Zurich, Department of Computer Science*, 260. DOI: 10.3929/ethz-a-006651937.

[16] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. 2020. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In *ACM CCS 2020*. ACM, 1769–1787. DOI: 10.1145/3372297.3423367.

[17] Ignacio Cascudo and Bernardo David. 2024. Publicly verifiable secret sharing over class groups and applications to DKG and YOSO. In *EUROCRYPT 2024* (LNCS 14655). Springer, 216–248. DOI: 10.1007/978-3-031-58740-5_8.

[18] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. 2020. Bandwidth-efficient threshold EC-DSA. In *PKC 2020* (LNCS 12111). Springer, 266–296. DOI: 10.1007/978-3-030-45388-6_10.

[19] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. 2023. Bandwidth-efficient threshold EC-DSA revisited: online/offline extensions, identifiable aborts proactive and adaptive security. *Theoretical Computer Science*, 939, 78–104, C. DOI: 10.1016/j.tcs.2022.10.016.

[20] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. 2019. Two-party ECDSA from hash proof systems and efficient instantiations. In *CRYPTO 2019* (LNCS 11694). Springer, 191–221. DOI: 10.1007/978-3-030-26954-8_7.

[21] Guilhem Castagnos and Fabien Laguillaumie. 2015. Linearly homomorphic encryption from DDH. In *CT-RSA 2015* (LNCS 9048). Springer, 487–505. DOI: 10.1007/978-3-319-16715-2_26.

[22] Guilhem Castagnos, Fabien Laguillaumie, and Ida Tucker. 2018. Practical fully secure unrestricted inner product functional encryption modulo p. In *ASIACRYPT 2018*. Springer, 733–764. DOI: 10.1007/978-3-030-03329-3_25.

[23] David Chaum and Torben Pryds Pedersen. 1992. Wallet databases with observers. In *CRYPTO '92* (LNCS 740). Springer, 89–105. DOI: 10.1007/3-540-48071-4_7.

[24] Arka Rai Choudhuri, Aarushi Goel, Matthew Green, Abhishek Jain, and Gabriel Kaptchuk. 2021. Fluid MPC: secure multiparty computation with dynamic participants. In *CRYPTO 2021* (LNCS 12826). Springer, 94–123. DOI: 10.1007/978-3-030-84245-1_4.

[25] Ran Cohen, Jack Doerner, Yashvanth Kondi, and Abhi Shelat. 2024. Secure multiparty computation with identifiable abort via vindicating release. In *CRYPTO 2024*. Springer, Cham, 36–73. DOI: 10.1007/978-3-031-68397-8_2.

[26] Ran Cohen and Yehuda Lindell. 2017. Fairness versus guaranteed output delivery in secure multiparty computation. *Journal of Cryptology*, 30, 4, 1157–1186. DOI: 10.1007/s00145-016-9245-5.

[27] Ivan Damgård, Thomas Pelle Jakobsen, Jesper Buus Nielsen, Jakob Illeborg Pagter, and Michael Bæksvang Østergaard. 2020. Fast threshold ECDSA with honest majority. In *SCN 2020* (LNCS 12238). Springer, 382–400. DOI: 10.1007/978-3-030-57990-6_19.

[28] Sourav Das, Zhuolun Xiang, and Ling Ren. 2021. Asynchronous data dissemination and its applications. In *ACM CCS 2021*. ACM, 2705–2721. DOI: 10.1145/3460120.3484808.

[29] Yi Deng, Shunli Ma, Xinxuan Zhang, Hailong Wang, Xuyang Song, and Xiang Xie. 2021. Promise Σ-protocol: how to construct efficient threshold ECDSA

from encryptions based on class groups. In *ASIACRYPT 2021* (LNCS 13093). Springer, 557–586. DOI: 10.1007/978-3-030-92068-5_19.
[30] Yvo Desmedt and Yair Frankel. 1990. Threshold cryptosystems. In *CRYPTO '89* (LNCS 435). Springer, 307–315. DOI: 10.1007/0-387-34805-0_28.
[31] 2021. Diogenes: lightweight scalable RSA modulus generation with a dishonest majority. In IEEE S&P 2021, 590–607. DOI: 10.1109/SP40001.2021.00025.
[32] Samuel Dobson, Steven Galbraith, and Benjamin Smith. 2022. Trustless unknown-order groups. *Mathematical Cryptology*, 1, 2, 25–39. https://journals.flvc.org/mathcryptology/article/view/130579.
[33] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. 2018. Secure two-party threshold ECDSA from ECDSA assumptions. In *IEEE S&P 2018*, 980–997. DOI: 10.1109/SP.2018.00036.
[34] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. 2019. Threshold ECDSA from ECDSA assumptions: the multiparty case. In *IEEE S&P 2019*, 1051–1066. DOI: 10.1109/SP.2019.00024.
[35] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. 2024. Threshold ECDSA in three rounds. In IEEE S&P 2024. IEEE Computer Society, 3053–3071. DOI: 10.1109/SP54263.2024.00178.
[36] Paul Feldman. 1987. A practical scheme for non-interactive verifiable secret sharing. In *IEEE SFCS 1987*, 427–438. DOI: 10.1109/SFCS.1987.4.
[37] Marc Fischlin. 2005. Communication-efficient non-interactive proofs of knowledge with online extractors. In *CRYPTO 2005* (LNCS 3621). Springer, 152–168. DOI: 10.1007/11535218_10.
[38] Offir Friedman, Avichai Marmor, Dolev Mutzari, Omer Sadika, Yehonatan C. Scaly, Yuval Spiizer, and Avishay Yanai. 2024. 2PC-MPC: emulating two party ECDSA in large-scale MPC. Cryptology ePrint Archive, Paper 2024/253. (2024).
[39] Adam Gągol, Jędrzej Kula, Damian Straszak, and Michał Świętek. 2020. Threshold ECDSA for decentralized asset custody. Cryptology ePrint Archive, Paper 2020/498. (2020).
[40] Rosario Gennaro and Steven Goldfeder. 2018. Fast multiparty threshold ECDSA with fast trustless setup. In *ACM CCS 2018*. ACM, 1179–1194. DOI: 10.1145/3243734.3243859.
[41] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. 2016. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In *ACNS 2016* (LNCS 9696). Springer, 156–174. DOI: 10.1007/978-3-319-39555-5_9.
[42] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. 1996. Robust threshold DSS signatures. In *EUROCRYPT '96* (LNCS 1070). Springer, 354–371. DOI: 10.1007/3-540-68339-9_31.
[43] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. 2007. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20, 1, 51–83. DOI: 10.1007/s00145-006-0347-3.
[44] Rosario Gennaro, Darren Leigh, Ravi Sundaram, and William Yerazunis. 2004. Batching Schnorr identification scheme with applications to privacy-preserving authorization and low-bandwidth communication devices. In *ASIACRYPT 2004* (LNCS 3329). Springer, 276–292. DOI: 10.1007/978-3-540-30539-2_20.
[45] Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. 2021. YOSO: you only speak once. Secure MPC with stateless ephemeral roles. In *CRYPTO 2021* (LNCS 12826). Springer, 64–93. DOI: 10.1007/978-3-030-84245-1_3.
[46] Jens Groth and Victor Shoup. 2022. Design and analysis of a distributed ECDSA signing service. Cryptology ePrint Archive, Paper 2022/506. (2022).
[47] Jens Groth and Victor Shoup. 2022. On the security of ECDSA with additive key derivation and presignatures. In *EUROCRYPT 2022* (LNCS 13275). Springer, 365–396. DOI: 10.1007/978-3-031-06944-4_13.
[48] Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. 2021. Aggregatable distributed key generation. In *EUROCRYPT 2021* (LNCS 12696). Springer, 147–176. DOI: 10.1007/978-3-030-77870-5_6.
[49] Aniket Kate, Easwar Vivek Mangipudi, Pratyay Mukherjee, Hamza Saleem, and Sri Aravinda Krishnan Thyagarajan. 2024. Non-interactive VSS using class groups and application to DKG. In *ACM CCS 2024*. ACM, 4286–4300. DOI: 10.1145/3658644.3670312.
[50] Jonathan Katz. 2024. Round optimal, fully secure distributed key generation. In *CRYPTO 2024* (LNCS 14926). Springer, 285–316. DOI: 10.1007/978-3-031-68394-7_10.
[51] Jonathan Katz and Antoine Urban. 2024. Honest-majority threshold ECDSA with batch generation of key-independent presignatures. Cryptology ePrint Archive, Paper 2024/2011. (2024).
[52] Sermin Kocaman and Younes Talibi Alaoui. 2023. Efficient secure two party ECDSA. In *IMACC 2023* (LNCS 14421). Springer, 161–180. DOI: 10.1007/978-3-031-47818-5_9.
[53] Brian Koziel, S. Dov Gordon, and Craig Gentry. 2024. Fast two-party threshold ECDSA with proactive security. In *ACM CCS 2024*. ACM, 1567–1580. DOI: 10.1145/3658644.3670387.
[54] Susan K. Langford. 1995. Threshold DSS signatures without a trusted party. In *CRYPTO '95* (LNCS 963). Springer, 397–409. DOI: 10.1007/3-540-44750-4_32.
[55] Yehuda Lindell. 2017. Fast secure two-party ECDSA signing. *Journal of Cryptology*, 34, 4. DOI: 10.1007/s00145-021-09409-9.
[56] Yehuda Lindell and Ariel Nof. 2018. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In *ACM CCS 2018*. ACM, 1837–1854. DOI: 10.1145/3243734.3243788.
[57] Philip MacKenzie and Michael K. Reiter. 2001. Two-party generation of DSA signatures. In *CRYPTO 2001* (LNCS 2139). Springer, 137–154. DOI: 10.1007/3-540-44647-8_8.
[58] Wafa Neji, Kaouther Blibech, and Narjes Ben Rajeb. 2016. Distributed key generation protocol with a new complaint management strategy. *Security and Communication Networks*, 9, 17, 4585–4595. DOI: 10.1002/sec.1651.
[59] Michaella Pettit. 2021. Efficient threshold-optimal ECDSA. In *CANS 2021* (LNCS 13099). Springer, 116–135. DOI: 10.1007/978-3-030-92548-2_7.
[60] C. P. Schnorr. 1990. Efficient identification and signatures for smart cards. In *CRYPTO '89*. Springer, 239–252. DOI: 10.1007/0-387-34805-0_22.
[61] Adi Shamir. 1979. How to share a secret. *Communications of the ACM*, 22, 11, 612–613. DOI: 10.1145/359168.359176.
[62] Binbin Tu, Yu Chen, Hongrui Cui, and Xianfang Wang. 2024. Fast two-party signature for upgrading ECDSA to two-party scenario easily. *Theoretical Computer Science*, 986. DOI: 10.1016/j.tcs.2023.114325.
[63] Dmytro Tymokhanov and Omer Shlomovits. 2021. Alpha-rays: key extraction attacks on threshold ECDSA implementations. Cryptology ePrint Archive, Paper 2021/1621. (2021).
[64] Harry W. H. Wong, Jack P. K. Ma, and Sherman S. M. Chow. 2024. Secure multiparty computation of threshold signatures made more efficient. In NDSS 2024. DOI: 10.14722/ndss.2024.24601.
[65] Harry W. H. Wong, Jack P. K. Ma, Hoover H. F. Yin, and Sherman S. M. Chow. 2023. Real threshold ECDSA. In NDSS 2023. DOI: 10.14722/ndss.2023.24817.
[66] Haiyang Xue, Man Ho Au, Mengling Liu, Kwan Yin Chan, Handong Cui, Xiang Xie, Tsz Hon Yuen, and Chengru Zhang. 2023. Efficient multiplicative-to-additive function from Joye-Libert cryptosystem and its application to threshold ECDSA. In *ACM CCS 2023*. ACM, 2974–2988. DOI: 10.1145/3576915.3616595.
[67] Haiyang Xue, Man Ho Au, Xiang Xie, Tsz Hon Yuen, and Handong Cui. 2021. Efficient online-friendly two-party ECDSA signature. In *ACM CCS 2021*. ACM, 558–573. DOI: 10.1145/3460120.3484803.
[68] Tsz Hon Yuen, Handong Cui, and Xiang Xie. 2021. Compact zero-knowledge proofs for threshold ECDSA with trustless setup. In *PKC 2021* (LNCS 12710). Springer, 481–511. DOI: 10.1007/978-3-030-75245-3_18.
[69] Guy Zyskind, Avishay Yanai, and Alex Pentland. 2024. Unstoppable wallets: chain-assisted threshold ECDSA and its applications. In *ACM ASIA-CCS 2024*. ACM, 1844–1860. DOI: 10.1145/3634737.3637657.

A INSTANTIATING THE ZERO-KNOWLEDGE PROOFS

For the relation $\mathcal{R} = \{(\mathsf{st}; \mathsf{wit})\}$, where st is the *statement* or public input and wit is the *witness* or secret input, a $\mathsf{NIZK}_{\mathcal{R}}$ proof π associated with a probabilistic algorithm Prove and a deterministic algorithm Verify satisfies

Completeness If $(\mathsf{st}; \mathsf{wit}) \in \mathcal{R}$ and $\pi \leftarrow \mathsf{Prove}(\mathsf{st}; \mathsf{wit})$, then $\mathsf{Verify}(\pi; \mathsf{st}) = 1$ with probability 1.

Computational soundness Without knowing $(\mathsf{st}; \mathsf{wit}) \in \mathcal{R}$, no efficient adversary can come up with π satisfying $\mathsf{Verify}(\pi; \mathsf{st}) = 1$ with non-negligible probability.

Statistical special zero knowledge A simulator that can program the Random Oracle can come up with π satisfying $\mathsf{Verify}(\pi; \mathsf{st}) = 1$ unless with negligible probability, even if no wit exists such that $(\mathsf{st}; \mathsf{wit}) \in \mathcal{R}$. The distribution of π is statistically close to real proofs.

Proofs with computational soundness are also called *arguments*, while some NIZKs satisfy a stronger property:

Straight-line extractability If $\mathsf{Verify}(\pi; \mathsf{st}) = 1$, then an extractor that can observe queries to the Random Oracle can retrieve wit such that $(\mathsf{st}; \mathsf{wit}) \in \mathcal{R}$.

Chaum–Pedersen proof of Discrete Log Equality. The Σ-protocol is folklore [23]. We apply a variant of the Fiat–Shamir heuristic, which we attribute to [15], that produces an equivalent, more compact

$\mathsf{NIZK}_{\mathsf{DLEQ}}$

Statement: $g_1, X_1, g_2, X_2 \in \mathbb{G}$
Witness: x
Claim: $X_1 = g_1^x \wedge X_2 = g_2^x$
Prove$(\mathsf{st}; \mathsf{wit}) \to \pi$:
$u \leftarrow\!\$\, \mathbb{Z}_q$; $U_1 \leftarrow g_1^u$; $U_2 \leftarrow g_2^u$; $e \leftarrow \mathsf{H}(\mathsf{st}, U_1, U_2)$;
$z \leftarrow u + ex \bmod q$; $\pi \leftarrow (e, z)$
Verify$(\mathsf{st}; \pi) \to \{0,1\}$:
$\hat{U}_1 \leftarrow g_1^z / X_1^e$; $\hat{U}_2 \leftarrow g_2^z / X_2^e$; Check: $e \stackrel{?}{=} \mathsf{H}(\mathsf{st}, \hat{U}_1, \hat{U}_2)$

Figure 9: Proof of discrete log equality in $\mathbb{G}$.

proof with only two $\mathbb{Z}_q$ elements (Figure 9). We use this more compact formulation for all our NIZKs. Since inversions over both the elliptic curve group and the class group are computationally free – only a change of sign is needed – there is no overhead in using this alternative Fiat–Shamir formulation in our case.

The NIZK simulator works for these proofs by sampling z's, deriving U's according to verification equations, and then programming the Random Oracle to output some random e.

Arguments in the class group. For relations where a secret is an exponent in both elliptic curve and class groups, we construct NIZK arguments based on the *promise* Σ-protocols of Deng et al. [29]. The key advantage of [29] over proofs in [20] using binary challenges is improved efficiency by an order of magnitude. Compared with arguments in [18, 68], the computational cost of [29] is similar, but the soundness of [29] does not depend on special assumptions over the class group such as Strong Root and Low Order.

The strategy above does not apply to the proof of honest CL threshold decryption, where the secret is an exponent exclusively in the unknown-order subgroup. We use the arguments of Braun et al. [14], the soundness of which depends on the Rough Order assumption. As our NIZKs are derived from public-coin Σ-protocols via the Fiat–Shamir transform, their *completeness*, *computational soundness*, and *statistical special zero knowledge* can be proven based on the properties of the underlying Σ-protocols.

Let $\mathcal{B} = 2^{\lambda_{\mathsf{st}}} \cdot q \cdot \tilde{s}$, where λ_{st} is a statistical security parameter that we set to 80 and $\tilde{s}$ is a rough upper bound of the order of $\mathfrak{G}^q$. This bound is used in sampling the first-move randomness and in checking the responses are small.

The $\mathsf{NIZK}_{\mathsf{PVSS}}$ proof (Figure 10) is from [49], and $\mathsf{NIZK}_{\mathsf{Aff}}$ (Figure 11) is our new construction. Each of them batches n EQ-relations into a proof of $O(1)$ size. We refer to [44, Figure 2] for the batching technique. For polynomial evaluations in the exponent, Horner's algorithm is recommended so that the computational cost is close to unbatched. The NIZK proofs used in RompSig-L are given in Figures 12, 13, 14, 15.

As an aside, the two $\mathsf{NIZK}_{\mathsf{PVSS}}$ proofs sent by each party in the first round of RompSig-Q presigning can use the same challenge γ hashed from the combined statement of two PVSS instances, so that prover computation can be approximately halved, and verifier computation can be reduced by 25%.

$\mathsf{NIZK}_{\mathsf{PVSS}}$

Statement: $\{\mathsf{pk}_j\}_j, R, \{E_j\}_j, g, \{A_k\}_k$
Witness: $\{s_j\}_j, r$
Claim: $R = g_q^r \wedge \forall j \left(g^{s_j} = \prod_k A_k^{j^k} \wedge E_j = f^{s_j} \mathsf{pk}_j^r\right)$
Prove$(\mathsf{st}; \mathsf{wit}) \to \pi$:
$u_1 \leftarrow\!\$\, [\mathcal{B}]$; $u_2 \leftarrow\!\$\, \mathbb{Z}_q$; $U_1 \leftarrow g_q^{u_1}$; $U_2 \leftarrow g^{u_2}$; $\gamma \leftarrow \mathsf{H}(\mathsf{st})$;
$U_3 \leftarrow f^{u_2} \left(\prod_j \mathsf{pk}_j^{\gamma^j}\right)^{u_1}$; $e \leftarrow \mathsf{H}(\gamma, U_1, U_2, U_3)$;
$z_1 \leftarrow u_1 + er$; $z_2 \leftarrow u_2 + e \sum_j s_j \cdot \gamma^j \bmod q$;
$\pi \leftarrow (e, z_1, z_2)$
Verify$(\mathsf{st}; \pi) \to \{0,1\}$:
Check: $z_1 < \mathcal{B}$, $z_2 \in \mathbb{Z}_q$; $\gamma \leftarrow \mathsf{H}(\mathsf{st})$;
$\hat{U}_1 \leftarrow g_q^{z_1} / R^e$; $\hat{U}_2 \leftarrow g^{z_2} / \left(\prod_k A_k^{\sum_j j^k \gamma^j}\right)^e$;
$\hat{U}_3 \leftarrow f^{z_2} \left(\prod_j \mathsf{pk}_j^{\gamma^j}\right)^{z_1} / \left(\prod_j E_j^{\gamma^j}\right)^e$;
Check: $e \stackrel{?}{=} \mathsf{H}(\gamma, \hat{U}_1, \hat{U}_2, \hat{U}_3)$

Figure 10: Proof of Publicly-Verifiable Secret Sharing.

$\mathsf{NIZK}_{\mathsf{Aff}}$

Statement: $\{c_j, \tilde{c}_j, M_j, \tilde{M}_j\}_j, h, g, A,$ satisfying:
$(\exists R, \{E_j\}_j, \tilde{R}, \{\tilde{E}_j\}_j)\ (\forall j)\ c_j = (R, E_j),\ \tilde{c}_j = (\tilde{R}, \tilde{E}_j)$
Witness: $a, \{b_j\}_j$
Claim: $A = g^a \wedge \forall j \left(\tilde{c}_j = a \otimes c_j \oplus (1, f^{b_j}) \wedge \tilde{M}_j = M_j^a h^{b_j}\right)$
Prove$(\mathsf{st}; \mathsf{wit}) \to \pi$:
$u_1 \leftarrow\!\$\, [\mathcal{B}]$; $u_2 \leftarrow\!\$\, \mathbb{Z}_q$; $\gamma \leftarrow \mathsf{H}(\mathsf{st})$; $U_1 \leftarrow g^{u_1}$; $U_2 \leftarrow R^{u_1}$;
$U_3 \leftarrow f^{u_2} \left(\prod_j E_j^{\gamma^j}\right)^{u_1}$; $U_4 \leftarrow h^{u_2} \left(\prod_j M_j^{\gamma^j}\right)^{u_1}$;
$e \leftarrow \mathsf{H}(\gamma, U_1, U_2, U_3, U_4)$; $z_1 \leftarrow u_1 + ea$;
$z_2 \leftarrow u_2 + e \sum_j b_j \cdot \gamma^j \bmod q$; $\pi \leftarrow (e, z_1, z_2)$
Verify$(\mathsf{st}; \pi) \to \{0,1\}$:
Check: $z_1 < \mathcal{B}$, $z_2 \in \mathbb{Z}_q$; $\gamma \leftarrow \mathsf{H}(\mathsf{st})$;
$\hat{U}_1 \leftarrow g^{z_1} / A^e$; $\hat{U}_2 \leftarrow R^{z_1} / \tilde{R}^e$;
$\hat{U}_3 \leftarrow f^{z_2} \left(\prod_j E_j^{\gamma^j}\right)^{z_1} / \left(\prod_j \tilde{E}_j^{\gamma^j}\right)^e$;
$\hat{U}_4 \leftarrow h^{z_2} \left(\prod_j M_j^{\gamma^j}\right)^{z_1} / \left(\prod_j \tilde{M}_j^{\gamma^j}\right)^e$;
Check: $e \stackrel{?}{=} \mathsf{H}(\gamma, \hat{U}_1, \hat{U}_2, \hat{U}_3, \hat{U}_4)$

Figure 11: Proof of consistent affine operations on multi-receiver ciphertext and on MACs with correct scalar.

Proof of knowledge of CL secret key. For our security proof to go through, the proof for $\mathcal{R}_{\mathsf{Key}}$ should satisfy *straight-line extractability*. The proof of knowledge for the same relation in [18] uses binary challenges to achieve special soundness. Combined with the Fischlin transform [37], this gives a straight-line extractable NIZK.

B SECURITY PROOF

In this section, we prove our security theorems. We first overview our methodology of proving unforgeability; namely, we build reductions to the enhanced existential unforgeability of single-party ECDSA. We then describe the simulations for RompSig-Q and RompSig-L, respectively. Finally we prove the robustness property.

$\mathsf{NIZK}_{\text{CL-DLEQ}}$

Statement: $g_1, X_1, g_2, X_2 \in \mathbb{G},\ \mathsf{pk},\ c = (R, E)$
Witness: x, r
Claim: $c = (g_q^r, f^x \mathsf{pk}^r) \wedge X_1 = g_1^x \wedge X_2 = g_2^x$
$\mathsf{Prove}(\mathsf{st}; \mathsf{wit}) \to \pi$:
$u_1 \leftarrow\!\$\ [\mathcal{B}];\ u_2 \leftarrow\!\$\ \mathbb{Z}_q;\ U_1 \leftarrow g_q^{u_1};\ U_2 \leftarrow f^{u_2}\mathsf{pk}^{u_1};$
$U_3 \leftarrow g_1^{u_2};\ U_4 \leftarrow g_2^{u_2};\ e \leftarrow \mathsf{H}(\mathsf{st}, U_1, U_2, U_3, U_4);$
$z_1 \leftarrow u_1 + er;\ z_2 \leftarrow u_2 + ex \bmod q;\ \pi \leftarrow (e, z_1, z_2)$
$\mathsf{Verify}(\mathsf{st}; \pi) \to \{0,1\}$:
Check: $z_1 < \mathcal{B},\ z_2 \in \mathbb{Z}_q;\ \hat{U}_1 \leftarrow g_q^{z_1}/R^e;$
$\hat{U}_2 \leftarrow f^{z_2}\mathsf{pk}^{z_1}/E^e;\ \hat{U}_3 \leftarrow g_1^{z_2}/X_1^e;\ \hat{U}_4 \leftarrow g_2^{z_2}/X_2^e;$
Check: $e \stackrel{?}{=} \mathsf{H}(\mathsf{st}, \hat{U}_1, \hat{U}_2, \hat{U}_3, \hat{U}_4)$

Figure 12: Proof of CL-encrypted 2-equal discrete log in $\mathbb{G}$.

$\mathsf{NIZK}_{\text{CL-EG}}$

Statement: $\mathsf{pk},\ \hat{c} = (C, \Lambda),\ Y,\ c = (E, \Gamma)$
Witness: $m, \hat{r}, r$
Claim: $\hat{c} = (g_q^{\hat{r}}, f^m \mathsf{pk}^{\hat{r}}) \wedge c = (g^r, g^m Y^r)$
$\mathsf{Prove}(\mathsf{st}; \mathsf{wit}) \to \pi$:
$u_1 \leftarrow\!\$\ [\mathcal{B}];\ u_2, u_3 \leftarrow\!\$\ \mathbb{Z}_q;\ U_1 \leftarrow g_q^{u_1};\ U_2 \leftarrow f^{u_2}\mathsf{pk}^{u_1};$
$U_3 \leftarrow g^{u_3};\ U_4 \leftarrow g^{u_2} Y^{u_3};\ e \leftarrow \mathsf{H}(\mathsf{st}, U_1, U_2, U_3, U_4);$
$z_1 \leftarrow u_1 + e\hat{r};\ z_2 \leftarrow u_2 + em \bmod q;$
$z_3 \leftarrow u_3 + er \bmod q;\ \pi \leftarrow (e, z_1, z_2, z_3)$
$\mathsf{Verify}(\mathsf{st}; \pi) \to \{0,1\}$:
Check: $z_1 < \mathcal{B},\ z_2, z_3 \in \mathbb{Z}_q;\ \hat{U}_1 \leftarrow g_q^{z_1}/C^e;$
$\hat{U}_2 \leftarrow f^{z_2}\mathsf{pk}^{z_1}/\Lambda^e;\ \hat{U}_3 \leftarrow g^{z_3}/E^e;\ \hat{U}_4 \leftarrow g^{z_2} Y^{z_3}/\Gamma^e;$
Check: $e \stackrel{?}{=} \mathsf{H}(\mathsf{st}, \hat{U}_1, \hat{U}_2, \hat{U}_3, \hat{U}_4)$

Figure 13: Proof of CL–ElGamal double encryption.

$\mathsf{NIZK}_{\text{CL-Scal2}}$

Statement: $\mathsf{pk}, c_a, g, A, c_1, \tilde{c}_1, c_2, \tilde{c}_2,$ satisfying: $c_a = (R, E)$,
$c_1 = (R_1, E_1), \tilde{c}_1 = (\tilde{R}_1, \tilde{E}_1), c_2 = (R_2, E_2), \tilde{c}_2 = (\tilde{R}_2, \tilde{E}_2)$
Witness: a, r
Claim: $c_a = (g_q^r, f^a \mathsf{pk}^r) \wedge A = g^a \wedge \tilde{c}_1 = c_1 \otimes a \wedge \tilde{c}_2 = c_2 \otimes a$
$\mathsf{Prove}(\mathsf{st}; \mathsf{wit}) \to \pi$:
$u_1, u_2 \leftarrow\!\$\ [\mathcal{B}];\ U_1 \leftarrow g_q^{u_1};\ U_2 \leftarrow f^{u_2}\mathsf{pk}^{u_1};\ U_3 \leftarrow g^{u_2};$
$U_4 \leftarrow R_1^{u_2};\ U_5 \leftarrow E_1^{u_2};\ U_6 \leftarrow R_2^{u_2};\ U_7 \leftarrow E_2^{u_2};$
$e \leftarrow \mathsf{H}(\mathsf{st}, U_1, U_2, U_3, U_4, U_5, U_6, U_7);$
$z_1 \leftarrow u_1 + er;\ z_2 \leftarrow u_2 + ea;\ \pi \leftarrow (e, z_1, z_2)$
$\mathsf{Verify}(\mathsf{st}; \pi) \to \{0,1\}$:
Check: $z_1, z_2 < \mathcal{B};\ \hat{U}_1 \leftarrow g_q^{z_1}/R^e;\ \hat{U}_2 \leftarrow f^{z_2}\mathsf{pk}^{z_1}/E^e;$
$\hat{U}_3 \leftarrow g^{z_2}/A^e;\ \hat{U}_4 \leftarrow R_1^{z_2}/\tilde{R}_1^e;\ \hat{U}_5 \leftarrow E_1^{z_2}/\tilde{E}_1^e;\ \hat{U}_6 \leftarrow R_2^{z_2}/\tilde{R}_2^e;$
$\hat{U}_7 \leftarrow E_2^{z_2}/\tilde{E}_2^e;$ Check: $e \stackrel{?}{=} \mathsf{H}(\mathsf{st}, U_1, U_2, U_3, U_4, U_5, U_6, U_7)$

Figure 14: Proof of CL-encrypted scalar on two ciphertexts.

$\mathsf{NIZK}_{\text{CGDL3}}$

Statement: $g_1, X_1, g_2, X_2, g_3, X_3 \in \mathfrak{G}^q$
Witness: x
Claim: $X_1 = g_1^x \wedge X_2 = g_2^x \wedge X_3 = g_3^x$
$\mathsf{Prove}(\mathsf{st}; \mathsf{wit}) \to \pi$:
$u \leftarrow\!\$\ [\mathcal{B}];\ U_1 \leftarrow g_1^u;\ U_2 \leftarrow g_2^u;\ U_3 \leftarrow g_3^u;$
$e \leftarrow \mathsf{H}(\mathsf{st}, U_1, U_2, U_3);\ z \leftarrow u + ex;\ \pi \leftarrow (e, z)$
$\mathsf{Verify}(\mathsf{st}; \pi) \to \{0,1\}$:
Check: $z < \mathcal{B};\ \hat{U}_1 \leftarrow g_1^z/X_1^e;\ \hat{U}_2 \leftarrow g_2^z/X_2^e;\ \hat{U}_3 \leftarrow g_3^z/X_3^e;$
Check: $c \stackrel{?}{=} \mathsf{H}(\mathsf{st}, \hat{U}_1, \hat{U}_2, \hat{U}_3)$

Figure 15: Proof of double threshold decryption with correct secret key share.

B.1 Unforgeability Proof Overview

In Figure 16 we define the enhanced existential unforgeability of single-party ECDSA [16]. The adversary is only given a verification key $\tilde{X}$ and access to a presignature oracle and a signature oracle. The adversary wins the experiment if he forges a valid signature under $\tilde{X}$ for a previously unqueried message. In [16] it is shown that the probability of the adversary winning this experiment is negligible, although the exact security is revisited in [47] and a security degradation that increases with the number of unused presignatures is uncovered. We remark that this issue is limited to a scenario where signing requests are sent by an adversary, and in this case no meaningful security is possible. To date, all threshold-optimal ECDSA schemes perform presigning in the same approach as ours. Anyway, we do consider incorporating the mitigations in [47] a future improvement.

We then prove the unforgeability of our threshold ECDSA schemes by constructing reductions to $\mathsf{Exp}_{\mathcal{A}}^{\text{ecdsa-e-euf}}$. Let $\mathcal{A}$ be the adversary corrupting $t-1$ parties in the threshold ECDSA scheme, and we show that if $\mathcal{A}$ forges signatures with non-negligible probability, then we can use it to build a forger $\mathcal{F}$ that forges signatures and wins in $\mathsf{Exp}_{\mathcal{A}}^{\text{ecdsa-e-euf}}$.

Similarly to prior game-based proofs [40, 19, 65, 64], we construct the reduction via simulation. Let Sim be a simulator that simulates the environment for $\mathcal{A}$. Without loss of generality, let $\mathcal{A}$ corrupt parties $P_1, \ldots, P_{t-1}$, and let the simulator Sim act on behalf of parties $P_t, \ldots, P_m$, where $t \le m \le n$. Sim also responds to the random oracle queries of $\mathcal{A}$. As we assume $\mathcal{A}$ is rushing, parties controlled by Sim have to speak first in each round.

The overall idea of our proof is as follows. Sim must embed the challenge verification key $\tilde{X}$ obtained in $\mathsf{Exp}_{\mathcal{A}}^{\text{ecdsa-e-euf}}$ as the verification key of the parties, by simulating the TKeyGen session. Then, the adversary will request signatures for messages that he adaptively chooses, and for each of them, Sim queries the presignature oracle and the signature oracle in $\mathsf{Exp}_{\mathcal{A}}^{\text{ecdsa-e-euf}}$ to obtain a signature (r, s), and simulates a TSign session that also outputs it. If the simulation succeeds, and the view of $\mathcal{A}$ in the simulation is indistinguishable from that in real executions of the threshold ECDSA scheme, then the adversary should output a valid forgery for an unsigned message at the same probability as in the real world, and if it happens, we can use this forgery to win in $\mathsf{Exp}_{\mathcal{A}}^{\text{ecdsa-e-euf}}$.

We describe the algorithm Sim, and then prove that the simulation succeeds and is indistinguishable from a real execution,

MAIN:

$\tilde{x} \leftarrow\$ \mathbb{Z}_q$; $\tilde{X} \leftarrow g^{\tilde{x}}$;
$Q_{\text{presign}}, Q_{\text{used}}, Q_{\text{sign}} \leftarrow \emptyset$
$(\mathtt{msg}^*, r^*, s^*) \leftarrow\$ \mathcal{A}^{O^{\text{presign}}, O^{\text{sign}}}(\tilde{X})$
if $(g^{\mathsf{H}(\mathtt{msg}^*)} \tilde{X}^{r^*})^{1/s^*}|_{x\text{-axis}} = r^*$
  $\wedge\ \mathtt{msg}^* \notin Q_{\text{sign}}$: **return** 1
return 0

Oracle O^{presign}:

$k \leftarrow\$ \mathbb{Z}_q$; $R \leftarrow g^k$
$Q_{\text{presign}} \leftarrow Q_{\text{presign}} \cup \{(R, k)\}$
return R

Oracle $O^{\text{sign}}(\mathtt{msg}, R)$:

if $\nexists k$ s.t. $(R, k) \in Q_{\text{presign}}$
  $\vee\ R \in Q_{\text{used}}$: **return** $\perp$
$Q_{\text{used}} \leftarrow Q_{\text{used}} \cup \{R\}$
$r \leftarrow R|_{x\text{-axis}}$
$s \leftarrow (\mathsf{H}(\mathtt{msg}) + r\tilde{x})/k$
$Q_{\text{sign}} \leftarrow Q_{\text{sign}} \cup \{\mathtt{msg}\}$
return (r, s)

Figure 16: $\mathsf{Exp}^{\text{ecdsa-e-euf}}_{\mathcal{A}}$, experiment of enhanced unforgeability of single-party ECDSA under chosen-message attacks.

unless with negligible probability, under some hardness assumptions. While in game-based security the simulation typically can be standalone, i.e., the simulator is allowed to rewind the environment, ours do not use rewinding. We extract secrets necessary for successful simulation by first extracting the CL secret keys of the adversary, then simply decrypting them from ciphertexts. To extract CL secret keys, Sim observes random oracle queries made by the adversary when computing the Fischlin proof of knowledge.

The hardness assumptions are as follows:

Decisional Diffie–Hellman (DDH) For any $g \in \mathbb{G}$ and uniformly random $a, b, c \in \mathbb{Z}_q$, $\mathcal{A}$ cannot distinguish between the tuples (g, g^a, g^b, g^{ab}) and (g, g^a, g^b, g^c) with non-negligible probability.

Hard Subgroup Membership (HSM) [12] For $r \leftarrow\$ \mathcal{D}_q$ and $m \in \mathbb{Z}_q$, $\mathcal{A}$ cannot distinguish between g_q^r and $f^m g_q^r$ with non-negligible probability.

B.2 Simulation of RompSig-Q

Let $C := \{1, \ldots, t-1\}$ be the indices of parties corrupted by $\mathcal{A}$, and $\mathcal{H} := \{t, \ldots, m\}$ be the indices of parties controlled by Sim.

Simulating Distributed Key Generation. Our forger $\mathcal{F}$ first initiates $\mathsf{Exp}^{\text{ecdsa-e-euf}}_{\mathcal{A}}$ and obtains the challenge verification key $\tilde{X}$. The simulator Sim then has to simulate a RompSig-Q TKeyGen session (Figure 4) that outputs $\tilde{X}$. Sim interacts with $\mathcal{A}$ as follows:

(1) Each $P_i,\ i \in \mathcal{H}$ honestly generates the CL key pair, broadcasts the public key, and proves knowledge of the secret key.
(2) Observe the queries to the random oracle, and use the straight-line extractor to retrieve the CL secret keys of corruped parties $\{\mathsf{sk}_j\}_{j \in C}$.
(3) Each $P_i,\ i \in \mathcal{H}$ honestly participates in JointPVSS with fresh randomness.
(4) Use $\{\mathsf{sk}_j\}_{j \in C \cup \mathcal{H}}$ to decrypt $\{x_j\}_{j \in C \cup \mathcal{H}}$ from $\{c_{x_j}\}_{j \in C \cup \mathcal{H}}$.
(5) Instead of opening X_i honestly, each $P_i,\ i \in \mathcal{H}$ simulates one such that the public key interpolates to $\tilde{X}$:

$$X_i^{L_{i,C\cup\{i\}}} \cdot \prod_{j \in C} (g^{x_j})^{L_{j,C\cup\{i\}}} = \tilde{X}$$
$$\iff X_i = (\tilde{X} / \prod_{j \in C} (g^{x_j})^{L_{j,C\cup\{i\}}})^{1/L_{i,C\cup\{i\}}}.$$

(6) Each $P_i,\ i \in \mathcal{H}$ simulates a $\mathsf{NIZK}_{\mathsf{DLEQ}}$ proof π_{x_i}, and broadcasts (X_i, π_{x_i}).

Simulating Presigning. When $\mathcal{A}$ requests a presignature, let our forger $\mathcal{F}$ query O^{presign} in $\mathsf{Exp}^{\text{ecdsa-e-euf}}_{\mathcal{A}}$ and obtain a presignature R, and let Sim simulate a presigning session (Figure 5) that outputs R as follows:

(1) Round 1: Each $P_i,\ i \in \mathcal{H}$ honestly participates in the two sessions of JointPVSS with fresh randomness.
(2) Using $\{\mathsf{sk}_j\}_{j \in C \cup \mathcal{H}}$, decrypt $\{k_j\}_{j \in C \cup \mathcal{H}}$ from $\{c_{k_j}\}_{j \in C \cup \mathcal{H}}$, and decrypt $\{\phi_j\}_{j \in C \cup \mathcal{H}}$ from $\{c_{\phi_j}\}_{j \in C \cup \mathcal{H}}$.
(3) Round 2: Instead of opening R_i honestly, each $P_i, i \in \mathcal{H}$ simulates one such that the presignature interpolates to R given by O^{presign}:

$$R_i^{L_{i,C\cup\{i\}}} \cdot \prod_{j \in C} (g^{k_j})^{L_{j,C\cup\{i\}}} = R$$
$$\iff R_i = (R / \prod_{j \in C} (g^{k_j})^{L_{j,C\cup\{i\}}})^{1/L_{i,C\cup\{i\}}}.$$

(4) Each $P_i,\ i \in \mathcal{H}$ simulates a $\mathsf{NIZK}_{\mathsf{DLEQ}}$ proof π_{R_i}.
(5) Each $P_i,\ i \in \mathcal{H}$ honestly performs MtA with fresh randomness, even though k_i and x_i do not align with R_i and X_i.
(6) Each $P_i,\ i \in \mathcal{H}$ decrypts $\{\alpha_{j,i}\}_{j \in C}$ and $\{\mu_{j,i}\}_{j \in C}$ from ciphertexts that are to be given to $\mathcal{A}$, using extracted CL secret keys.
(7) Each $P_i,\ i \in \mathcal{H}$ simulates $\mathsf{NIZK}_{\mathsf{Aff}}$ proofs $\pi_{\alpha_{*,i}}$, $\pi_{\mu_{*,i}}$, assembles the message $\mathsf{presignfin}_i$ and broadcasts it.
(8) As prescribed in the protocol, each $P_i,\ i \in \mathcal{H}$ checks the NIZK proofs, decrypts $\{\alpha_{i,j}\}_j$ and $\{\mu_{i,j}\}_j$ from ciphertexts sent to itself, and derives R and r which should be what is given by O^{presign}, but defers the rest of "presigning local finalization" to the online phase.

Simulating Online Signing. When $\mathcal{A}$ requests a signature for a message $\mathtt{msg}$, let our forger $\mathcal{F}$ query O^{sign} with $\mathtt{msg}$ and a previously unused presignature R in $\mathsf{Exp}^{\text{ecdsa-e-euf}}_{\mathcal{A}}$, and obtain a valid signature (r, s), then let Sim simulate the adversary's view in the online signing phase (Figure 5) that consumes the presignature R and outputs s, as follows:

(1) Randomly sample $\hat{k} \leftarrow\$ \mathbb{Z}_q$, and let $\hat{x} = (\hat{k}s - \mathsf{H}(\mathtt{msg}))/r$, though they are not the real nonce and signing key and do not agree with R and X.
(2) Each $P_i,\ i \in \mathcal{H}$ ignores what have been obtained in JointPVSS, and instead calculates $\hat{k}_i$ and $\hat{x}_i$ such that, along with $\{k_j\}_{j \in C}$ and $\{x_j\}_{j \in C}$ known by $\mathcal{A}$, they interpolate to the newly sampled $\hat{k}$ and $\hat{x}$:

$$\hat{k}_i \cdot L_{i,C\cup\{i\}} + \textstyle\sum_{j \in C} k_j \cdot L_{j,C\cup\{i\}} = \hat{k}$$
$$\iff \hat{k}_i = (\hat{k} - (\textstyle\sum_{j \in C} k_j \cdot L_{j,C\cup\{i\}}))/L_{i,C\cup\{i\}};$$
$$\hat{x}_i \cdot L_{i,C\cup\{i\}} + \textstyle\sum_{j \in C} x_j \cdot L_{j,C\cup\{i\}} = \hat{x}$$
$$\iff \hat{x}_i = (\hat{x} - (\textstyle\sum_{j \in C} x_j \cdot L_{j,C\cup\{i\}}))/L_{i,C\cup\{i\}}.$$

(3) Each $P_i,\ i \in \mathcal{H}$ computes $\{\theta_{i,j}\}_j$, shares of 0, as prescribed. Then, instead of following the protocol, let

$$\{u_{i,j}\}_j = \{\alpha_{i,j} + \hat{k}_i\phi_j - \alpha_{j,i} + \theta_{i,j}\}_{j\in\mathcal{P}\setminus\{i\}} \cup \{u_{i,i} = \hat{k}_i\phi_i + \theta_{i,i}\}.$$

(4) Each $P_i,\ i \in \mathcal{H}$ computes $\{m_{i,j}\}_j$, shares of $m = \mathsf{H}(\mathtt{msg})$, as prescribed, then let

$$\{w_{i,j}\}_j = \{m_{i,j}\phi_i + r(\mu_{i,j} + \hat{x}_i\phi_j - \mu_{j,i})\}_{j\in\mathcal{P}\setminus\{i\}} \cup \{w_{i,i} = m_{i,i}\phi_i + r\phi_i\hat{x}_i\}.$$

In a sense, these two steps equivocate $\beta_{j,i}$ and $\nu_{j,i}$, which P_i has generated locally and held as MtA fragments, so that the partial products hit new values needed to yield the signature.

(5) Round 3: Each $P_i,\ i \in \mathcal{H}$ computes $U_i = \Phi^{\hat{k}_i}$, $V_i = \Phi^{\hat{x}_i}$, simulates π_{U_i}, π_{V_i}, assembles and broadcasts $\mathsf{onlinesign}_i$. It is easy to check that the final message passes verification.

We analyze the above simulation. The simulation of TKeyGen aborts if Sim fails to extract the CL secret keys, or if the adversary has queried the random oracle on special inputs and Sim cannot program it to simulate the NIZK proofs. To achieve the latter, the adversary has to guess the randomness used by Sim, and can only succeed with negligible probability. The simulation of a TSign session aborts if Sim fails to decrypt a value from ciphertexts, or if Sim fails to simulate a NIZK proof. This can only happen with negligible probability. We can thus conclude that the above simulation succeeds unless with negligible probability.

To finish the proof we must argue that the simulation of a signing session is indistinguishable from a real execution. First, the simulated NIZKs are indistinguishable from real due to the statistical zero knowledge property. Second, we must apply the same reasoning as in [19, Lemma 4] to prove that the k_i, x_i generated in encrypted form in JointPVSS and used in MtA does not make it detectable that different values are used in their place later. This relies on the HSM assumption and the property of CL encryption being *smooth*; we refer the reader to [19] for details. Then, by DDH we have that the differences induced in step (5) of the DKG simulation, step (3) of the presigning simulation, and step (5) of the online-signing simulation are indistinguishable. Finally, the difference induced in steps (2, 3) of the online-signing simulation is indistinguishable because they align with a real execution that produces the signature (r, s) and the values follow the same distribution. This completes the proof.

B.3 Simulation of RompSig-L

Let $C := \{1, \ldots, t-1\}$ be the indices of parties corrupted by $\mathcal{A}$, and $\mathcal{H} := \{t, \ldots, m\}$ be the indices of parties controlled by Sim.

Simulating Distributed Key Generation. Our forger $\mathcal{F}$ first initiates $\mathsf{Exp}^{\text{ecdsa-e-euf}}_{\mathcal{A}}$ and obtains the challenge verification key $\tilde{X}$. The simulator Sim then has to simulate a RompSig-Q TKeyGen session (Figure 6) that outputs $\tilde{X}$. Sim interacts with $\mathcal{A}$ as follows:

(1) Each $P_i,\ i \in \mathcal{H}$ honestly participates in the DKG for threshold CL encryption.
(2) Observe the queries to the random oracle, and use the straightline extractor to retrieve the CL secret key shares of corruped parties $\{\mathsf{sk}_j\}_{j\in C}$.
(3) Each $P_i,\ i \in \mathcal{H}$ honestly participates in the two JointPVSS sessions.
(4) Using $\{\mathsf{sk}_j\}_{j\in C\cup\mathcal{H}}$, decrypt $\{x_j\}_{j\in C\cup\mathcal{H}}$ from $\{c_{x_j}\}_{j\in C\cup\mathcal{H}}$, and decrypt $\{y_j\}_{j\in C\cup\mathcal{H}}$ from $\{c_{y_j}\}_{j\in C\cup\mathcal{H}}$.
(5) Instead of opening X_i honestly, each $P_i,\ i \in \mathcal{H}$ simulates one such that the public key interpolates to $\tilde{X}$:

$$X_i^{L_{i,C\cup\{i\}}} \cdot \textstyle\prod_{j\in C}(g^{x_j})^{L_{j,C\cup\{i\}}} = \tilde{X}$$
$$\iff X_i = (\tilde{X}/\textstyle\prod_{j\in C}(g^{x_j})^{L_{j,C\cup\{i\}}})^{1/L_{i,C\cup\{i\}}}.$$

(6) Each $P_i,\ i \in \mathcal{H}$ gives a dummy encryption of 0 under pk as c_i, simulates a $\mathsf{NIZK}_{\mathsf{CL\text{-}DLEQ}}$ proof π_{x_i}, and broadcasts (X_i, c_i, π_{x_i}).
(7) Derive output as prescribed by the protocol.

Simulating Presigning. When $\mathcal{A}$ requests a presignature, let our forger $\mathcal{F}$ query O^{presign} in $\mathsf{Exp}^{\text{ecdsa-e-euf}}_{\mathcal{A}}$ and obtain a presignature R, and let Sim simulate a presigning session (Figure 7) that outputs R as follows:

(1) Round 1: Each $P_i,\ i \in \mathcal{H}$ honestly participates in the "nonce proposal" round with fresh randomness.
(2) Round 2: Derive $c_R = (E, \Gamma)$ as normal, but instead of honestly giving the partial decryption Ω_i, each $P_i,\ i \in \mathcal{H}$ simulates one such that R, given by O^{presign}, is threshold-decrypted from it:

$$R = \Gamma/(\Omega_i^{L_{i,C\cup\{i\}}} \cdot \textstyle\prod_{j\in C}(E^{y_j})^{L_{j,C\cup\{i\}}})$$
$$\iff \Omega_i = (\Gamma/(R \cdot \textstyle\prod_{j\in C}(E^{y_j})^{L_{j,C\cup\{i\}}}))^{1/L_{i,C\cup\{i\}}}.$$

(3) Each $P_i,\ i \in \mathcal{H}$ simulates a $\mathsf{NIZK}_{\mathsf{DLEQ}}$ proof π_{Ω_i}, and performs the rest of presigning as prescribed, but defers calculating U_i to the online phase.

Simulating Online Signing. When $\mathcal{A}$ requests a signature for a message msg, let our forger $\mathcal{F}$ query O^{sign} with msg and a previously unused presignature R in $\mathsf{Exp}^{\text{ecdsa-e-euf}}_{\mathcal{A}}$, and obtain a valid signature (r, s), then let Sim simulate the adversary's view in the online signing phase (Figure 7) that consumes the presignature R and outputs s, as follows:

(1) Randomly sample $\hat{u} \leftarrow\!\!\$\ \mathbb{Z}_q$, and let $\hat{w} = s\hat{u}$.
(2) Instead of honestly giving the partial decryption share U_i, each $P_i,\ i \in \mathcal{H}$ simulates one such that $\hat{u}$ is decrypted from c_u:

$$\Lambda_u^{\Delta^3} = f^{\Delta^3\hat{u}} U_i^{\Delta\cdot\mathcal{L}_{i,C\cup\{i\}}} \textstyle\prod_{j\in C} C_u^{\mathsf{sk}_j\cdot\Delta\cdot\mathcal{L}_{j,C\cup\{i\}}}$$
$$\iff U_i = \frac{\Lambda_u^{\Delta^2/\mathcal{L}_{i,C\cup\{i\}}} f^{-\Delta^2\hat{u}/\mathcal{L}_{i,C\cup\{i\}}}}{\prod_{j\in C} C_u^{\mathsf{sk}_j\cdot(\mathcal{L}_{j,C\cup\{i\}}/\mathcal{L}_{i,C\cup\{i\}})}}.$$

(3) Similarly, let $\hat{w}$ be threshold-decrypted from c_w:

$$W_i = \frac{\Lambda_w^{\Delta^2/\mathcal{L}_{i,C\cup\{i\}}} f^{-\Delta^2\hat{w}/\mathcal{L}_{i,C\cup\{i\}}}}{\prod_{j\in C} C_w^{\mathsf{sk}_j\cdot(\mathcal{L}_{j,C\cup\{i\}}/\mathcal{L}_{i,C\cup\{i\}})}}.$$

It can be checked that all exponents are in $\mathbb{Z}$, as sk_j is divisible by Δ.

(4) Round 3: Each $P_i,\ i \in \mathcal{H}$ simulates a $\mathsf{NIZK}_{\mathsf{CGDL3}}$ proof π_{d_i}, then assembles and broadcasts $\mathsf{onlinesign}_i$.

Similarly to prior analysis, we can conclude that the simulation terminates unless with negligible probability.

To finish the proof we must argue that the simulation of a signing session is indistinguishable from a real execution. First, the

simulated NIZKs are indistinguishable from real due to the statistical zero knowledge property. Second, by the IND-CPA security of CL encryption and DDH, we have that the differences induced in steps (5, 6) of the DKG simulation and step (2) of the presigning simulation are indistinguishable. Finally, the difference induced in steps (2, 3) of the online-signing simulation is indistinguishable by the HSM assumption. This completes the proof.

B.4 Robustness

Because we assume a broadcast channel, the views of different parties are consistent, and any misbehaving party are identified and disqualified by all honest parties, hence the set of qualified parties is identically defined for every honest party. This is ensured by the soundness of NIZKs in all three rounds of RompSig-L and RompSig-Q, and the information-theoretical security of the consistency check in the last round of RompSig-Q. Thus the t-robustness of both protocols simply follows from the fact that, if t or more parties properly behave, then the protocol can proceed even when some parties drop out midway, and the output is correct. The reader can find more detailed proofs of robustness in [65, 64].

C THE LEGACY BLUEPRINT

Compared with the newer approach [1, 35], earlier works on threshold ECDSA follow a less round-efficient blueprint. In those schemes signing involves generating $R = g^{k^{-1}}$ and $s = k(m + xr)$, where x is the signing key, m is the hash of the message, and $r = R|_{x\text{-axis}}$. The protocols proceed as follows:

(1) Distributively generate secret-shared nonce $[\![k]\!]$ and helper variable $[\![\gamma]\!]$.
(2) Multiply to get secret-shared products $[\![k\gamma]\!]$ and $[\![kx]\!]$, and open $\Gamma = g^{\gamma}$.
(3) Open $k\gamma$, and locally compute $R = \Gamma^{(k\gamma)^{-1}} = g^{k^{-1}}$.
(4) Open $k(m + xr)$, a secret sharing of which can be homomorphically transformed from $[\![k]\!]$ and $[\![kx]\!]$.

Each step requires at least one round, and the lowest round count under this blueprint is four [16, 64]. To our knowledge, this blueprint dates back to [42].