Threshold ECDSA in Two Rounds

Yingjie Lyu*
yingjielyu@mail.sdu.edu.cn
Shandong University

Zengpeng Li*
lizengpeng@sdu.edu.cn
Shandong University

Hong-Sheng Zhou
hszhou@vcu.edu
Virginia Commonwealth University

Xudong Deng*
xudongdenng@gmail.com
Shandong University

ABSTRACT

We propose the first two-round multi-party signing protocol for the Elliptic Curve Digital Signature Algorithm (ECDSA) in the threshold-optimal setting, reducing the number of rounds by one compared to the state of the art (Doerner et al., S&P '24). We also resolve the security issue of presigning pointed out by Groth and Shoup (Eurocrypt '22), evading a security loss that increases with the number of pre-released, unused presignatures, for the first time among threshold-optimal schemes.

Our construction builds on Non-Interactive Multiplication (NIM), a notion proposed by Boyle et al. (PKC '25), which allows parties to evaluate multiplications on secret-shared values in one round. In particular, we use the construction of Abram et al. (Eurocrypt '24) instantiated with class groups. The setup is minimal and transparent, consisting of only two class-group generators. The signing protocol is efficient in bandwidth, with a message size of 1.9 KiB at 128-bit security, and has competitive computational performance.

CCS CONCEPTS

• **Security and privacy → Digital signatures**.

KEYWORDS

ECDSA, signatures, threshold cryptography, blockchain, cryptocurrencies

ACM Reference Format:
Yingjie Lyu, Zengpeng Li, Hong-Sheng Zhou, and Xudong Deng. 2025. Threshold ECDSA in Two Rounds. In *Proceedings of the 2025 ACM Conference on Computer and Communications Security (CCS '25).* ACM, New York, NY, USA, 14 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

In a (t, n) threshold signature scheme [28], the signing key is secret-shared among n parties, and to issue a signature, a subset of at least t parties have to collaboratively interact. Threshold signatures are especially useful in cryptocurrency wallets for the extra security they provide. To forge signatures, an adversary has to corrupt t parties, which is much harder than in the single-party case. In recent years, threshold ECDSA has received extensive attention [35, 34, 43, 29, 15, 16, 33, 27, 53, 2, 8, 17, 50, 10, 52, 49, 30, 32, 25, 26, 46, 36], as ECDSA is a long-standing NIST standard and is widely adopted across major blockchains.

The Quest for Fewer Rounds. We consider the threshold-optimal setting, where the threshold t can be chosen arbitrarily from 2 to n. In this most general case, the earliest practical constructions by Gennaro and Goldfeder [34] and Lindell and Nof [43] require eight rounds of interaction. This was later improved to four rounds by Canetti et al. [15], and more recently, a three-round protocol was introduced by Doerner et al. [30], which remains the state of the art. In contrast, simple and efficient *two-round* threshold Schnorr schemes are already known [41, 6], naturally prompting the question:

> *Is a practical two-round threshold ECDSA protocol possible?*

Due to the inherent complexity of ECDSA, this question has remained unresolved. Recent advances by Boneh et al. [9] and Adjedj et al. [4] have demonstrated two-round protocols in the two-party setting, but extending these results to the general multi-party case remains an open challenge.

Handling Many Presigning Sessions Securely. Many threshold ECDSA schemes since [15, 26] leverage preprocessing to minimize signing latency. The signing protocol is split into an offline *presigning* phase – run before the message to be signed is known – and an online phase. Ideally, presigning should dominate the total cost, and the online phase should be fast and *non-interactive*: each signer sends one protocol message upon receiving the signing input, and the messages are publicly combined into the signature output.

However, an analysis by Groth and Shoup [37] shows that security degrades when too many presigning sessions are run in advance. Specifically, for each prematurely revealed R value (presignature), if the adversary can find two message hashes whose ratio hits one of polynomially many special values, it can forge a signature. As a result, the security that existing threshold ECDSA schemes with presigning can possibly achieve is worse than that of plain ECDSA. An additional assumption on the hash function must be introduced, which is not needed for plain ECDSA. Moreover, this weakness completely breaks any scheme that both performs presigning and admits signing raw message hashes at the same time.

Some mitigations have been incorporated in two-party [4] and honest-majority [36] schemes. However, in the threshold-optimal

*School of Cyber Science and Technology, Shandong University; State Key Laboratory of Cryptography and Digital Economy Security, Shandong University; Quan Cheng Laboratory

CCS '25, October 13–17, 2025, Taipei, Taiwan

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

setting, existing schemes can only address this issue by limiting the number of presigning sessions, resulting in an unsatisfactory trade-off between security and efficiency. The full version of [30] warns that "presigning should only be used in practice by those who understand and accept the implications and risks associated with it." The performance cost of giving up presigning is high, and it would be preferable if one can implement presigning without such concerns.

Our Results. We present the first two-round threshold ECDSA protocol in the threshold-optimal, multi-party setting. This provides an affirmative answer to the motivating question, which is open since Doerner et al. [30] and recently highlighted by Boneh et al. [9]. Moreover, our protocol is the first in the threshold-optimal setting that is immune to the attack surface revealed in [37], reclaiming the full efficiency of presigning.

1.1 Technical Overview

We begin with essential background, followed by a high-level overview of our solution. We then discuss the efficiency of our scheme in comparison to prior work.

ECDSA. Let $\mathbb{G}$ be an elliptic curve group generated by g of prime order q. Let $x \in \mathbb{Z}_q$ be the signing key, and $X = g^x$ be the verification key. To sign a message, the SHA-2 hash of which is $m \in \mathbb{Z}_q$, one chooses a random nonce $k \in \mathbb{Z}_q$, computes $R = g^k$, and sets r to be the x-coordinate of R, i.e., $r := R|_{x\text{-axis}}$. The signature is (r, σ) where $\sigma = k^{-1}(m + rx) \bmod q$.

Baseline Threshold ECDSA. In threshold ECDSA schemes, both x and k are secret-shared among parties and cannot be known by anyone (since knowing k allows recovery of x). Computing σ thus requires multi-party computation of inversion and multiplication on shared secrets. This is more complex than in threshold Schnorr signatures, where secrets are simply linearly combined.

Our starting point is the scheme of Doerner et al. [30]. Let each party i hold an additive share x_i of the secret key $x = x_1 + \cdots + x_t$ across a quorum of t parties. Signing proceeds in three rounds:

- Round 1: Each party i samples random k_i and γ_i, commits to g^{k_i}, and initiates Oblivious Linear Evaluation (OLE) with all others using input γ_i.
- Round 2: Party i responds to each OLE instance, so each ordered pair (i, j) obtains additive shares of $k_i\gamma_j$ and $x_i\gamma_j$. Parties each decommit g^{k_i}, making $R = g^k$ and $r = R|_{x\text{-axis}}$ public, where $k = \sum_i k_i$. Each party locally aggregates to obtain additive shares $(k\gamma)_i$ and $(x\gamma)_i$ of products $k\gamma$ and $x\gamma$, where $\gamma = \sum_i \gamma_i$.
- Round 3: Given message hash m, each party sends $m\gamma_i + r(x\gamma)_i$ and $(k\gamma)_i$. Combining them gives $\gamma(m+rx)$ and $k\gamma$, then dividing yields $\sigma = k^{-1}(m + rx)$.

Many schemes use OLE to share multiplication results. Typically, OLE is realized in two rounds using Homomorphic Encryption or Oblivious Transfer. In such a protocol, Bob inputs b in round one; Alice inputs a in round two; the output satisfies $c + d = ab$, with c and d held by Alice and Bob, respectively.

Non-Interactive Multiplication. Towards two-round threshold signing, we leverage a key tool stemming from Homomorphic Secret Sharing (HSS). Abram et al. [3] proposed an elegant way for two parties to evaluate one multiplication on additively secret-shared values *in one round*. This scheme is initially branded as Bilinear HSS, and also fits a later refined notion of Non-Interactive Multiplication (NIM) by Boyle et al. [12]. The underlying technique relies on a Distributed Discrete Logarithm mechanism, instantiated over Paillier groups [44] or ideal class groups [1], among others. The construction is surprisingly simple. It brings bandwidth savings, and its computational cost is on par with OLE from homomorphic encryption. Though proposed in a semi-honest setting, it can be upgraded to malicious security using proofs of knowledge.

However, a new tool realizing multiplication with one fewer round does not make two-round threshold ECDSA immediately possible. All prior schemes require a commitment to g^{k_i} in the first round, the equivocation of which allows the simulation to make R hit a designated value. If we collapse the first two rounds of [30] into one, removing the commitment, the resulting scheme is likely insecure, because a rushing adversary would be able to bias the distribution of R. Thus, provable security of the two-round design on the horizon becomes the main challenge.

Presignature Re-randomization to the Rescue. A key insight in [37] is that re-randomizing the presignature at signing time restores security to the level of plain ECDSA. This approach is echoed in recent two-round threshold Schnorr schemes [41, 6].

Another valuable lesson is that threshold ECDSA can be built from different assumptions modeling single-party ECDSA with different "modes of operation." Several such assumptions [15, 37, 4] have been analyzed in the Generic Group Model (GGM). While GGM is a strong idealized model, such idealization is known to be required for ECDSA [38], and we can draw useful comparative conclusions from these analyses.

Adjedj et al. [4] extend [37] by allowing an adversary to tweak the nonce multiplicatively before it is properly re-randomized additively. Their analysis shows only a constant-factor security loss compared to plain ECDSA, which is still significantly better than presigning without re-randomization [15]. Inspired by their idea, we are finally able to construct a provably-secure two-round scheme.

Our Construction. We sketch our two-round signing protocol:

- Round 1: Each party i samples random k_i, γ_i, publishes g^{k_i}, and inputs k_i, x_i, and γ_i into NIM. Each pair (i, j) obtains additive shares of $k_i\gamma_j$ and $x_i\gamma_j$. After the interaction, each party holds additive shares of $k\gamma$ and $x\gamma$, where $k = \sum_i k_i$ and $\gamma = \sum_i \gamma_i$. The value g^k is publicly known.
- Round 2: As the message to be signed is known, the nonce is re-randomized from k to $zk + y$, where z, y are hashes derived from the transcript. Also, $R = g^{zk+y}$ and $r = R|_{x\text{-axis}}$ are known. Each party i sends $m\gamma_i + r(x\gamma)_i$ and $z(k\gamma)_i + y\gamma_i$. Combining them gives $\gamma(m + rx)$ and $\gamma(zk + y)$, then dividing yields $\sigma = (zk + y)^{-1}(m + rx)$.

Provided with the secrets of corrupted parties, which are extracted from their proofs of knowledge, the simulator can program the random oracles producing the re-randomizers z and y in a suitable way, so that a reduction to the assumption analyzed in [4] is possible, assuming the security of the building blocks.

Efficiency. Our scheme outperforms the state of the art in terms of bandwidth efficiency (table 2). Each party communicates 1.9 KiB in the presigning phase and 64 bytes in the online phase, at the 128-bit security level. It only uses 4% as much communication as the scheme of Doerner et al. [30], and 10% as much as the scheme of Canetti et al. [15], in peer-to-peer networks. If parties are connected via centralized servers, our advantage increases further, as each party's outbound communication is constant for ours but linear in the number of signers for most other schemes in this setting.

We implement our scheme and compare it with some prior schemes in computational cost. The one of Doerner et al. [30] is very lightweight in computation, as it builds on OT extension and for the most part only performs symmetric-cryptographic operations. However, even though our presigning is about 10x heavier in computation than [30] and takes about 200 milliseconds CPU time for 3 parties, our scheme enjoys a fast online phase taking less than 1 millisecond of computation. Since our protocol is about 4x faster than [15], which has seen particular industrial interest, we regard our proposed scheme to be competitive in practical performance. As class groups find increasing use in cryptography, we expect that our scheme will benefit from future optimizations in this area.

1.2 Related Work

We compare the basic features of existing practical threshold ECDSA schemes in the threshold-optimal setting in table 1.

Table 1: Overview of threshold-optimal ECDSA schemes.

Scheme	Main Tool	Rounds	Note
GG18 [34]	Paillier	8	BC
LN18 [43]	Paillier / OT	8	original ver.
	OT	5	updated ver.
DKLs19 [29]	OT	$\lceil \log t \rceil + 6$	BC
CGGMP20 [15]	Paillier	4 or 7	BC, IA
CCLST20 [16]	CL	8	BC
CCLST23 [17]	CL	7	BC, IA
WMYC23 [50]	CL	7	BC, IA, robust
DKLs24 [30]	OT	3	
WMC24 [49]	threshold CL	4	BC, IA, robust
CDKS24 [22]	OT	7	BC, IA
This work	class group NIM	2	

IA = Identifiable Abort. BC = Requires broadcast channel during signing (if IA is not required, the cheaper *echo broadcast* can be used instead). OT = Oblivious Transfer. CL = Castagnos–Laguillaumie encryption. PCG = Pseudorandom Correlated Generator. NIM = Non-Interactive Multiplication. Round count of LN18 is based on the updated full version.

Gennaro and Goldfeder [34] and Lindell and Nof [43] proposed the first practical threshold-optimal ECDSA schemes. Both constructions use two-party secure computation subprotocols, based on Paillier encryption or Oblivious Transfer (OT), to convert a product of shared secrets to additive shares.

Canetti et al. [15] extend [34] to support identifiable abort, i.e., identification of cheating parties in the case of signing abort, using NIZK arguments. Their protocol takes four rounds, one of which is a broadcast. In fact, broadcasts are necessary for identifiable abort with a dishonest majority [23]. To avoid broadcast overhead, we assume no broadcasts during signing, and our scheme does not have identifiable abort.

Castagnos et al. [16] use the Castagnos–Laguillaumie (CL) encryption scheme [19] from class groups in place of Pailler to achieve lower bandwidth and avoid range proofs used in [34]. They later extend it to support identifiable abort [17]. Wong et al. [50] utilize Distributed Key Generation to generate Shamir secret-shared nonces, so that faults do not always abort signing. Later, Wong et al. [49] construct threshold ECDSA from threshold CL encryption [14]. This revives the idea of Gennaro et al. [35] using threshold Paillier, which is impractical due to a highly expensive setup [21]. The drawback of [49] is higher latency in the online phase.

Abram et al. [2] proposed a low-bandwidth construction using a Pseudorandom Correlated Generator (PCG). Their signing protocol consumes the least bandwidth among all schemes when amortized over large batches, but only supports full threshold, i.e., $t = n$.

Observing from [2] that the two multiplications in [43] can be computed in parallel instead of sequentially to reduce presigning rounds, Doerner et al. [30] proposed a three-round construction, subsuming their earlier scheme taking more rounds [29]. Using OT extension, their protocol is light on computation, but requires higher bandwidth. Cohen et al. [22] extend it to a 7-round protocol with identifiable abort.

Damgård et al. [26], Groth and Shoup [36], Katz and Urban [40], and Kondi and Ravi [42] study threshold ECDSA in the honest-majority setting. The scheme in [36] is the only one with mitigations to the attack surface analyzed in [37] among honest-majority and threshold-optimal ones. The drawback of these honest-majority schemes is that threshold signing requires at least twice as many participants as are needed to simply reconstruct the secret key.

We refer the reader to [30] and the updated full version of [15] for information on other works.

1.3 Paper Organization

The rest of this paper is organized as follows. In section 2, we first recall ECDSA, then define the syntax and security of threshold signature schemes. Next, we introduce the cryptographic building blocks, namely, Non-Interactive Multiplication (NIM) and Non-Interactive Zero-Knowledge Arguments of Knowledge (NIZKAoK). Then, we outline the assumption on single-party ECDSA introduced in [4]. In section 3, we describe our two-round threshold ECDSA scheme using the building blocks and prove its security. In section 4, we outline how we instantiate the building blocks based on class groups. In section 5, we present performance metrics and analyze the efficiency of our proposed scheme.

2 PRELIMINARIES

General Notation. Let κ and κ_{st} denote the computational and statistical security parameters, respectively. Let $\mathsf{negl}(\kappa)$ denote a negligible function. We use ":=" for deterministic assignment, "$\leftarrow$" for assignment from a randomized algorithm, and "$\leftarrow\$$" for uniform random sampling. Groups are written in multiplicative notation. The integer range $[m..n]$ represents $\{m, m+1, \ldots, n\}$, and $[n]$ means $[1..n]$. We use PPT to denote probabilistic polynomial time.

2.1 ECDSA

Let $\mathbb{G}$ denote the elliptic curve group generated by g of prime order q, and let $\mathbb{Z}_q$ denote the field $\mathbb{Z}/q\mathbb{Z}$ of integers modulo q. Let $\mathsf{H}_{\mathsf{sig}} : \{0,1\}^* \to \mathbb{Z}_q$ be a cryptographic hash function.

DEFINITION 1 (ECDSA). *The Elliptic Curve Digital Signature Algorithm* ECDSA = (Setup, KeyGen, Sign, Verify) *is defined as follows:*

- $\mathsf{Setup}(1^\kappa) \to \mathsf{pp}$*: On input the security parameter, the setup algorithm returns public parameters* $\mathsf{pp} = ((\mathbb{G}, q, g), \mathsf{H}_{\mathsf{sig}})$*, which will be implicitly given as input to other algorithms.*
- $\mathsf{KeyGen}(\mathsf{pp}) \to (X, x)$*: On input the public parameters, the key generation algorithm samples a signing key* $x \leftarrow\!\!\$\ \mathbb{Z}_q$*, computes the verification key* $X := g^x$*, and returns the key pair* (X, x)*.*
- $\mathsf{Sign}(x, \mathsf{msg}) \to (r, \sigma)$*: On input the signing key* x *and the message* msg*, the signing algorithm computes* $m := \mathsf{H}_{\mathsf{sig}}(\mathsf{msg})$*, samples a nonce* $k \leftarrow\!\!\$\ \mathbb{Z}_q$*, computes* $R := g^k$*, its* x*-coordinate* $r := R|_{x\text{-axis}}$*, and* $\sigma := k^{-1}(m + rx) \bmod q$*, and returns the signature* (r, σ)*.*
- $\mathsf{Verify}(X, \mathsf{msg}, (r, \sigma)) \to \{0,1\}$*: On input the verification key* X*, the message* msg*, and the signature* (r, σ)*, the verification algorithm computes* $m := \mathsf{H}_{\mathsf{sig}}(\mathsf{msg})$*, computes* $\hat{R} := (g^m X^r)^{1/\sigma}$*, and returns* 1 *if* $\hat{R}|_{x\text{-axis}} = r$ *and* 0 *otherwise.*

2.2 Defining Threshold Signatures

We define the syntax of threshold signature schemes following [24].

DEFINITION 2 (TWO-ROUND THRESHOLD SIGNATURES). *A two-round threshold signature scheme* TS = (Setup, KeyGen, Presign, Sign, Combine, Verify) *has the following syntax:*

- $\mathsf{Setup}(1^\kappa) \to \mathsf{pp}$*: On input the security parameter, the setup algorithm returns public parameters* pp*, which will be implicitly given as input to other algorithms.*
- $\mathsf{KeyGen}(t, n) \to (\mathsf{pk}, \{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in[n]})$*: On input the threshold* t *and the number of parties* n*, the probabilistic key generation algorithm outputs a verification key* pk *that represents the entirety of the* n *parties, the set* $\{\mathsf{pk}_i\}_{i\in[n]}$ *of public keys for each party, and the set* $\{\mathsf{sk}_i\}_{i\in[n]}$ *of secret keys for each party. Each* sk_i *is privately held by party* i*, and* $(\mathsf{pk}, \{\mathsf{pk}_i\}_{i\in[n]})$ *is made public.*
- (Presign, Sign, Combine)*: These three algorithms make up the signing protocol. Let* $\mathbf{P} \subseteq [n]$ *be a quorum of* t *or more parties. For each signing session, each party indexed by* $i \in \mathbf{P}$ *runs the following two algorithms:*
  - $\mathsf{Presign}(i) \to (\mathsf{pm}_i^{(1)}, \mathsf{pst}_i)$*: The presigning algorithm returns the party's round-1 protocol message* $\mathsf{pm}_i^{(1)}$ *and presigning secret state* pst_i*.*
  - $\mathsf{Sign}(i, \mathsf{msg}, \{\mathsf{pm}_j^{(1)}\}_{j\in\mathbf{P}}, \mathsf{pst}_i) \to \mathsf{pm}_i^{(2)}$*: On input the message* msg *to be signed, the round-1 protocol messages* $\{\mathsf{pm}_j^{(1)}\}_{j\in\mathbf{P}}$ *from parties in the quorum, and the party's presigning secret state* pst_i*, the signing algorithm returns the party's round-2 protocol message* $\mathsf{pm}_i^{(2)}$*.*

  Then, any entity can run the following algorithm:
  - $\mathsf{Combine}(\mathsf{pk}, \mathsf{msg}, \{(\mathsf{pm}_j^{(1)}, \mathsf{pm}_j^{(2)})\}_{j\in\mathbf{P}}) \to \mathsf{sig}$*: On input the verification key* pk*, the message* msg *to be signed, and the protocol messages* $\{(\mathsf{pm}_j^{(1)}, \mathsf{pm}_j^{(2)})\}_{j\in\mathbf{P}}$ *of parties in the quorum in both rounds, the deterministic combining algorithm returns a signature* sig*.*
- $\mathsf{Verify}(\mathsf{pk}, \mathsf{msg}, \mathsf{sig}) \to \{0,1\}$*: On input the verification key* pk*, the message* msg*, and the signature* sig*, the verification algorithm returns* 1 *if the signature is valid for the message* msg *and the verification key* pk*, and* 0 *otherwise.*

Correctness can be intuitively defined: for all $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa)$ and $t \le n$, after TS.KeyGen is run, for every quorum $\mathbf{P} \subseteq [n]$, $|\mathbf{P}| \ge t$ and every message msg, if each party indexed by $i \in \mathbf{P}$ honestly participates in the two rounds of signing and the protocol messages are delivered properly, then the signature output by TS.Combine always passes verification.

DEFINITION 3 (STATIC UNFORGEABILITY). *Let* TS *be a two-round threshold signature scheme. For a PPT adversary* $\mathcal{A}$*, we consider the following experiment* $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{TSUF}}$*:*

(1) Initialization Phase. *Let* $\mathsf{pp} \leftarrow \mathsf{TS.Setup}(1^\kappa)$*. Initialize two lists* Q_{state}*,* Q_{sign} *as empty. On input the number of signers* n*, the threshold* t*, and the corruption set* $\mathbf{C}$ *from* $\mathcal{A}$*, check if* $t \le n$*,* $\mathbf{C} \subsetneq [n]$*, and* $|\mathbf{C}| < t$*. If so, run* $(\mathsf{pk}, \{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in[n]}) \leftarrow \mathsf{TS.KeyGen}(t, n)$*, and define the honest set* $\mathbf{H} := [n] \setminus \mathbf{C}$*; else, return* $\perp$*.*

(2) Signing Phase. *The adversary* $\mathcal{A}$ *has full control over any corrupted party indexed by* $i \in \mathbf{C}$*. It knows the secret keys* $\{\mathsf{sk}_i\}_{i\in\mathbf{C}}$ *of corrupted parties. Furthermore, it can query the following two oracles to engage honest parties in the signing protocol:*
- Oracle PRESIGN: *On input* (i, sid) *from* $\mathcal{A}$*, where* i *is a party index and* sid *is a session identifier, check if* $i \in \mathbf{H}$ *and no item* $(i, \mathsf{sid}, \cdot)$ *exists in* Q_{state}*. If so, let* $(\mathsf{pm}_i^{(1)}, \mathsf{pst}_i) \leftarrow \mathsf{TS.Presign}(i)$*, return the protocol message* $\mathsf{pm}_i^{(1)}$ *to* $\mathcal{A}$*, and append the secret state* $(i, \mathsf{sid}, \mathsf{pst}_i)$ *to* Q_{state}*. Else, return* $\perp$*.*
- Oracle SIGN: *On input* $(i, \mathsf{sid}, \mathbf{P}, \mathsf{msg}, \{\mathsf{pm}_j^{(1)}\}_{j\in\mathbf{P}})$ *from* $\mathcal{A}$*, check if* $i \in \mathbf{H}$*, there exists an item* $(i, \mathsf{sid}, \mathsf{pst}_i)$ *in* Q_{state}*, and either* $(\mathsf{sid}, \mathsf{msg}) \in Q_{\mathsf{sign}}$ *or no item* $(\mathsf{sid}, \cdot)$ *exists in* Q_{sign}*. If so, return the party's round-2 protocol message* $\mathsf{pm}_i^{(2)} \leftarrow \mathsf{TS.Sign}(i, \mathsf{msg}, \{\mathsf{pm}_j^{(1)}\}_{j\in\mathbf{P}}, \mathsf{pst}_i)$ *to* $\mathcal{A}$*, and append* $(\mathsf{sid}, \mathsf{msg})$ *to* Q_{sign} *if it does not already exist. Else, return* $\perp$*.*

(3) Outcome: *If* $\mathcal{A}$ *outputs* $(\mathsf{msg}, \mathsf{sig})$ *such that no item* $(\cdot, \mathsf{msg})$ *exists in* Q_{sign} *and* $\mathsf{TS.Verify}(\mathsf{pk}, \mathsf{msg}, \mathsf{sig}) = 1$*, then the experiment returns* 1*. Otherwise, it returns* 0*.*

We say TS *satisfies existential unforgeability under chosen message attack with static corruption (or, static unforgeability) if, for any PPT adversary* $\mathcal{A}$*, the probability that the experiment* $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{TSUF}}$ *returns* 1 *is negligible in* κ*.*

In the unforgeability game above, we do not make any assumptions about the underlying network. The network can be thought of as fully controlled by the adversary, who relays protocol messages with arbitrary timing and order. Note that we do not assume broadcasts; if an adversary sends inconsistent protocol messages to different parties, a valid signature will not be produced, and the session will abort but with no further consequences. The adversary can also initiate concurrent signing sessions. If the adversary outputs a valid forgery for a unsigned message, then it wins the experiment. By 'unsigned' we mean the message has not been queried to any honest party; this corresponds to TS-UF-0 in the hierarchy of definitions in [6].

2.3 Non-Interactive Multiplication

We define the syntax of Non-Interactive Multiplication following Boyle et al. [12], with minor adjustments. Concrete instantiations of this primitive can be found in [3]. We provide details on a class-group instantiation in section 4.

Definition 4 (Non-Interactive Multiplication). *Let $\mathbb{Z}_q$ be the field of integers modulo q. A Non-Interactive Multiplication (NIM) scheme* $\mathsf{NIM} = (\mathsf{Setup}, (\mathsf{Encode}_\rho, \mathsf{Decode}_\rho)_{\rho \in \{A,B\}})$ *has the following syntax:*

- $\mathsf{Setup}(1^\kappa, q) \to \mathsf{crs}$*: On input the security parameter and the prime q, the setup algorithm returns a Common Reference String* crs.
- $\mathsf{Encode}_\rho(\mathsf{crs}, v) \to (\mathsf{pe}_\rho, \mathsf{st}_\rho)$*: On input* crs *and* $v \in \mathbb{Z}_q$*, the encoding algorithm, randomized and parameterized by a role* $\rho \in \{A, B\}$*, outputs a public encoding* pe_ρ *and a secret state* st_ρ.
- $\mathsf{Decode}_\rho(\mathsf{crs}, \mathsf{pe}_{\bar\rho}, \mathsf{st}_\rho) \to z_\rho$*: On input* crs*, a public encoding* $\mathsf{pe}_{\bar\rho}$ *from a different role,*[1] *and a secret state* st_ρ*, the decoding algorithm, deterministic and parameterized by a role* $\rho \in \{A, B\}$*, outputs a share* $z_\rho \in \mathbb{Z}_q$.

We require the following properties to hold:

Correctness: *The scheme is correct if, for all* $x, y \in \mathbb{Z}_q$,

$$\Pr\left[z_A + z_B = xy \;\middle|\; \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\kappa) \\ (\mathsf{pe}_A, \mathsf{st}_A) \leftarrow \mathsf{Encode}_A(\mathsf{crs}, x) \\ (\mathsf{pe}_B, \mathsf{st}_B) \leftarrow \mathsf{Encode}_B(\mathsf{crs}, y) \\ z_A := \mathsf{Decode}(\mathsf{crs}, \mathsf{pe}_B, \mathsf{st}_A) \\ z_B := \mathsf{Decode}(\mathsf{crs}, \mathsf{pe}_A, \mathsf{st}_B) \end{array} \right] \geq 1 - \mathsf{negl}(\kappa).$$

Input Privacy: *The scheme is input-private if, for all PPT adversary* $\mathcal{A}$ *and role* $\rho \in \{A, B\}$*, the public encoding hides the message:*

$$\Pr\left[b^* = b \;\middle|\; \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\kappa) \\ (v_0, v_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{crs}) \\ b \leftarrow\$ \{0, 1\} \\ (\mathsf{pe}_\rho, \mathsf{st}_\rho) \leftarrow \mathsf{Encode}_\rho(\mathsf{crs}, v_b) \\ b^* \leftarrow \mathcal{A}(\mathsf{st}, \mathsf{pe}_\rho) \end{array} \right] \leq \frac{1}{2} + \mathsf{negl}(\kappa).$$

2.4 NIZK Arguments of Knowledge

We define Non-Interactive Zero-Knowledge Arguments of Knowledge in the random oracle model.

Definition 5 (Non-Interactive Zero-Knowledge Arguments of Knowledge). *Let* $\mathcal{L}$ *be an NP language with associated relation* $\mathcal{R}_\mathcal{L}$*. Let* H *be a random oracle. A Non-Interactive Zero-Knowledge Argument of Knowledge scheme* $\mathsf{NIZKAoK} = (\mathsf{Setup}, \mathsf{Prove}^\mathsf{H}, \mathsf{Verify}^\mathsf{H})$ *has the following syntax:*

- $\mathsf{Setup}(1^\kappa) \to \mathsf{pp}$*: On input the security parameter, the setup algorithm returns public parameters* pp*, which will be implicitly given as input to other algorithms.*
- $\mathsf{Prove}^\mathsf{H}(\mathsf{stmt}, \mathsf{wit}) \to \pi$*: On input a statement* stmt *and a witness* wit *such that* $(\mathsf{stmt}, \mathsf{wit}) \in \mathcal{R}_\mathcal{L}$*, the prover outputs a proof* π.
- $\mathsf{Verify}^\mathsf{H}(\mathsf{stmt}, \pi) \to \{0, 1\}$*: On input a statement* stmt *and proof* π*, the verifier outputs* 1 *for accept or* 0 *for reject.*

We require the following properties to hold:

Completeness: *The scheme is complete if, for all* $(\mathsf{stmt}, \mathsf{wit}) \in \mathcal{R}_\mathcal{L}$,

$$\Pr\left[\mathsf{Verify}^\mathsf{H}(\mathsf{stmt}, \pi) = 1 \;\middle|\; \pi \leftarrow \mathsf{Prove}^\mathsf{H}(\mathsf{stmt}, \mathsf{wit})\right] \geq 1 - \mathsf{negl}(\kappa).$$

[1] The notation $\bar\rho$ denotes the role that is not ρ; i.e., $\bar{A} = B$ and $\bar{B} = A$.

Zero Knowledge: *The scheme is zero-knowledge if, for all PPT* $\mathcal{A}$*, there exists a simulator* Sim^H *such that for all* $(\mathsf{stmt}, \mathsf{wit}) \in \mathcal{R}_\mathcal{L}$,

$$\Big| \Pr\left[\mathcal{A}^\mathsf{H}(\mathsf{stmt}, \pi) = 1 \;\middle|\; \pi \leftarrow \mathsf{Prove}^\mathsf{H}(\mathsf{stmt}, \mathsf{wit})\right] - \Pr\left[\mathcal{A}^\mathsf{H}(\mathsf{stmt}, \pi) = 1 \;\middle|\; \pi \leftarrow \mathsf{Sim}^\mathsf{H}(\mathsf{stmt})\right] \Big| \leq \mathsf{negl}(\kappa).$$

Knowledge Extractability: *The scheme is knowledge-extractable if, for all PPT* $\mathcal{A}$*, there exists an extractor* $\mathsf{Extract}^\mathcal{A}$ *such that*

$$\Pr\left[\begin{array}{l} \mathsf{Verify}^\mathsf{H}(\mathsf{stmt}, \pi^*) = 1 \\ \wedge\ (\mathsf{stmt}, \mathsf{wit}^*) \notin \mathcal{R}_\mathcal{L} \end{array} \;\middle|\; \begin{array}{l} (\mathsf{stmt}, \pi^*) \leftarrow \mathcal{A}^\mathsf{H}(\mathsf{pp}) \\ \mathsf{wit}^* \leftarrow \mathsf{Extract}^\mathcal{A}(\mathsf{stmt}, \pi^*) \end{array} \right] \leq \mathsf{negl}(\kappa).$$

2.5 Doubly-Enhanced Existential Unforgeability of ECDSA

Below, we recall the computational assumption on the Doubly-Enhanced Existential Unforgeability of single-party ECDSA from Adjedj et al. [4].

Definition 6 (Doubly-Enhanced Existential Unforgeability of ECDSA [4]). *For a PPT adversary* $\mathcal{A}$*, we consider the following experiment* $\mathsf{Exp}^{\mathsf{DEUF}}_\mathcal{A}$*:*

(1) Initialization Phase. *The challenger runs* $\mathsf{pp} \leftarrow \mathsf{ECDSA.Setup}(1^\kappa)$ *and* $(X, x) \leftarrow \mathsf{ECDSA.KeyGen}(\mathsf{pp})$*, sends the verification key X to* $\mathcal{A}$*, and initializes three query sets* Q_{presign}*,* Q_{tweak}*, and* Q_{sign} *as empty.*

(2) Signing Phase. *The challenger answers* $\mathcal{A}$*'s queries to the three oracles as follows:*
   - Oracle Presign: *Sample a nonce* $k \leftarrow\$ \mathbb{Z}_q$ *and return* $R := g^k$*. Append* (R, k) *to* Q_{presign}.
   - Oracle Tweak: *On input* (R, z, m) *from* $\mathcal{A}$*, check if there is an item* $(R, k) \in Q_{\mathsf{presign}}$ *but no* $(R, z, m, \cdot) \in Q_{\mathsf{tweak}}$*. If so, return* $\mu \leftarrow\$ \mathbb{Z}_q$*, and append* (R, z, m, μ) *to* Q_{tweak}*. Else, return* $\bot$.
   - Oracle Sign: *On input* (R, z, m, μ) *from* $\mathcal{A}$*, check if there is an item* $(R, k) \in Q_{\mathsf{presign}}$ *and* $(R, z, m, \mu) \in Q_{\mathsf{tweak}}$*. If so, compute* $r := g^{zk+\mu}|_{x\text{-axis}}$*, return* $\sigma := (zk + \mu)^{-1}(m + rx)$*, append m to* Q_{sign}*, and delete* (R, k) *from* Q_{presign}*. Else, return* $\bot$.

(3) Outcome. *If* $\mathcal{A}$ *outputs* $(\mathsf{msg}, (r, \sigma))$ *such that* $\mathsf{H}_{\mathsf{sig}}(\mathsf{msg}) \notin Q_{\mathsf{sign}}$ *and* $\mathsf{ECDSA.Verify}(X, \mathsf{msg}, (r, \sigma)) = 1$*, then the experiment returns* 1*. Otherwise, it returns* 0.

We say that the Doubly-Enhanced Existential Unforgeability of ECDSA holds if, for any PPT adversary $\mathcal{A}$*, the probability that the experiment* $\mathsf{Exp}^{\mathsf{DEUF}}_\mathcal{A}$ *returns* 1 *is negligible in* κ.

The experiment defined in the assumption above is a variant of the standard EUF-CMA game, with some features related to presigning. In the experiment $\mathsf{Exp}^{\mathsf{DEUF}}_\mathcal{A}$, for each presignature $R = g^k$ provided by the challenger, the adversary can tweak it with a multiplicative factor z of its choice. For each tuple (R, z, m) queried, where m is a message hash, the challenger returns a modifier μ chosen uniformly at random. Before the adversary decides which (R, z, m) to actually use for signing, it can query the Tweak oracle arbitrarily many times. For the signature queried at last, the challenger uses $zk + \mu$ as the nonce.

The experiment $\mathsf{Exp}^{\mathsf{DEUF}}_\mathcal{A}$ is an extension of the security experiment of ECDSA with re-randomized presignatures analyzed by Groth and Shoup [37, section 8], which is shown in the generic group model (GGM) [47] to be as secure as plain ECDSA without presigning. In the latter, the adversary is not allowed to tweak the presignature, or equivalently, z always equals 1; neither can $\mathcal{A}$ query modifiers with different message hashes for a single presignature. The added flexibility in $\mathsf{Exp}^{\mathsf{DEUF}}_{\mathcal{A}}$ allows us to use random oracles to derive re-randomizers in the protocol design. According to the analysis in the GGM by Adjedj et al. [4, theorem 2.1], this only incurs a constant security loss of 2^{-5}.

Despite what the terms suggest, the Doubly-Enhanced existential unforgeability is a weaker assumption than the Enhanced existential unforgeability [15, appendix E] underlying existing threshold ECDSA schemes with presigning. The latter assumes extra features on the hash function $\mathsf{H}_{\mathsf{sig}}$ and has worse security. In that security experiment, the presignature is never re-randomized. For further information, we refer the reader to [37].

3 TWO-ROUND THRESHOLD ECDSA

In this section, we first describe our proposed two-round threshold ECDSA scheme, and then prove it secure assuming the security properties of the building blocks in the random oracle model.

3.1 Construction

We begin with a high-level explanation. First, the public parameters for ECDSA and the CRS for NIM are set up, and two hash functions to be used to re-randomize the presignature are given in the setup phase. Then, KeyGen generates the Shamir secret-shared signing key and the corresponding public verification key. In practice this is done via a Distributed Key Generation (DKG) subprotocol, but to simplify presentation, we use a trusted dealer here. Additionally, each party publishes a NIM public encoding (under role B) of each secret-key share, with a NIZKAoK attesting that it is well-formed.

Each signing session consists of a presigning round and a signing round. In the presigning round, each party i sends to other parties a public encoding of a random γ_i under role A, a public encoding of a random k_i under role B, and g^{k_i}; in addition, each party computes a NIZKAoK to prove the message is well-formed. After receiving each other's message, the parties can use NIM decoding to obtain additive shares of products $k\gamma$ and $x\gamma$, where $k = \sum_i k_i$ and $\gamma = \sum_i \gamma_i$. While this is done in the presigning phase, we put these steps in the Sign procedure to fit the defined syntax.

After the message msg to be signed is known, two hashes z, y are publicly derived from the message and the transcript of interaction. The presignature is re-randomized from g^k to $R = (g^k)^z g^y$. Each party i submits its shares of $w = m\gamma + rx\gamma$ and $u = y\gamma + zk\gamma$ to finalize signing, where $r = R|_{x\text{-axis}}$ and $m = \mathsf{H}_{\mathsf{sig}}(\mathsf{msg})$.

In line with other works on threshld ECDSA, we assume authenticated communication. This can be realized by having each party sign each protocol message, binding it to a session identifier. Thus, the protocol implicitly requires a separate Public Key Infrastructure (PKI) and an external mechanism to produce session identifiers. They can be implemented using readily available resources, and we do not explicitly specify them here.

Threshold ECDSA Protocol

Setup: On input the security parameter 1^κ, the setup algorithm does:

1. Run $((\mathbb{G}, q, g), \mathsf{H}_{\mathsf{sig}}) \leftarrow \mathsf{ECDSA.Setup}(1^\kappa)$;
2. Choose two hash functions $\mathsf{H}_1, \mathsf{H}_2 : \{0,1\}^* \to \mathbb{Z}_q$;
3. Run $\mathsf{crs} \leftarrow \mathsf{NIM.Setup}(1^\kappa)$;
4. Return $\mathsf{pp} := ((\mathbb{G}, q, g), \mathsf{H}_{\mathsf{sig}}, \mathsf{H}_1, \mathsf{H}_2, \mathsf{crs})$.

KeyGen: On input the threshold t and number of parties n, the key generation subprotocol first does:

1. Sample a random polynomial of degree $t-1$ over $\mathbb{Z}_q$: $f(Z) = a_0 + a_1 Z + \cdots + a_{t-1} Z^{t-1}$;
2. Set $X := g^{a_0}$ as the verification key;
3. For each party indexed by $i \in [n]$, set $x_i := f(i)$ as its secret-key share, and set $X_i := g^{x_i}$ as the corresponding public-key share;
4. Send each x_i to party i privately, and make $(X, \{X_i\}_{i\in[n]})$ public.

Then, each party i does the following:

1. Compute $(\mathsf{pe}_{x,i}, \mathsf{st}_{x,i}) \leftarrow \mathsf{NIM.Encode}_B(\mathsf{crs}, x_i)$;
2. Compute $\pi_{x,i}$, a NIZKAoK that $\mathsf{pe}_{x,i}$ and X_i use a consistent x_i;
3. Broadcast $(\mathsf{pe}_{x,i}, \pi_{x,i})$, and store $\mathsf{st}_{x,i}$;
4. Abort if any other party broadcasts a NIZKAoK that fails verification.

Finally, the key generation subprotocol outputs the verification key $\mathsf{pk} := X$, the parties' public keys $\{\mathsf{pk}_i\}_{i\in[n]}$ where $\mathsf{pk}_i := (X_i, \mathsf{pe}_{x,i})$, and the parties' secret keys $\{\mathsf{sk}_i\}_{i\in[n]}$ where $\mathsf{sk}_i := (x_i, \mathsf{st}_{x,i})$.

Presign: On receiving a presignature request, each party i does:

1. Sample $k_i, \gamma_i \leftarrow\$ \mathbb{Z}_q$;
2. Compute:
   - $K_i := g^{k_i}$,
   - $\Gamma_i := g^{\gamma_i}$,
   - $(\mathsf{pe}_{k,i}, \mathsf{st}_{k,i}) \leftarrow \mathsf{NIM.Encode}_B(\mathsf{crs}, k_i)$,
   - $(\mathsf{pe}_{\gamma,i}, \mathsf{st}_{\gamma,i}) \leftarrow \mathsf{NIM.Encode}_A(\mathsf{crs}, \gamma_i)$;
3. Compute π_i, a NIZKAoK proving that $\mathsf{pe}_{k,i}$ and K_i use a consistent k_i and that $\mathsf{pe}_{\gamma,i}$ and Γ_i use a consistent γ_i.

The presigning algorithm outputs the round-1 protocol message $\mathsf{pm}_i^{(1)} := (K_i, \Gamma_i, \mathsf{pe}_{k,i}, \mathsf{pe}_{\gamma,i}, \pi_i)$ and the secret state $\mathsf{pst}_i := (k_i, \gamma_i, \mathsf{st}_{k,i}, \mathsf{st}_{\gamma,i})$ of party i. It is assumed that each party i stores its secret state pst_i securely and sends the protocol message $\mathsf{pm}_i^{(1)}$ to other parties authenticated and bound to a session identifier.

Sign: On receiving the presigning protocol messages $\{\mathsf{pm}_j^{(1)}\}_{j\in\mathbf{P}}$ from parties in the quorum $\mathbf{P}$ where $\mathbf{P} \subseteq [n]$, $|\mathbf{P}| \geq t$ and a signing request for the message msg, each party indexed by $i \in \mathbf{P}$ does:

1. Set $K := \prod_{i\in\mathbf{P}} K_i$;
2. For each $j \in \mathbf{P}\backslash\{i\}$:
   - Parse $\mathsf{pm}_j^{(1)}$ as $(K_j, \Gamma_j, \mathsf{pe}_{k,j}, \mathsf{pe}_{\gamma,j}, \pi_j)$;
   - Abort if the NIZKAoK π_j fails verification;
   - Compute:
     - $\alpha_{i,j} := \mathsf{NIM.Decode}_B(\mathsf{crs}, \mathsf{pe}_{\gamma,j}, \mathsf{st}_{k,i})$,
     - $\beta_{j,i} := \mathsf{NIM.Decode}_A(\mathsf{crs}, \mathsf{pe}_{k,j}, \mathsf{st}_{\gamma,i})$,
     - $\mu_{i,j} := \lambda_{i,\mathbf{P}} \cdot \mathsf{NIM.Decode}_B(\mathsf{crs}, \mathsf{pe}_{\gamma,j}, \mathsf{st}_{x,i})$,

– $v_{j,i} := \lambda_{j,\mathbf{P}} \cdot \mathsf{NIM.Decode}_A(\mathsf{crs}, \mathsf{pe}_{x,j}, \mathsf{st}_{\gamma,i})$.

Here, $\lambda_{i,\mathbf{P}} := \prod_{j \in \mathbf{P}\setminus\{i\}} j \cdot (j-i)^{-1} \bmod q$ is the i-th Lagrange coefficient w.r.t. the quorum. The steps above can be performed before msg is known.

(3) Compute:
- $m := \mathsf{H}_{\mathsf{sig}}(\mathsf{msg})$,
- $z := \mathsf{H}_1(X, \mathsf{msg}, \{\mathsf{pm}_i^{(1)}\}_{i\in\mathbf{P}})$,
- $y := \mathsf{H}_2(z)$;

(4) Set $R := K^z g^y$, and $r := R|_{x\text{-axis}}$;

(5) Set $w_i := m\gamma_i + r(\lambda_{i,\mathbf{P}} x_i \gamma_i + \sum_{j\in\mathbf{P}\setminus\{i\}} (\mu_{i,j} + \nu_{j,i}))$;

(6) Set $u_i := y\gamma_i + z(k_i\gamma_i + \sum_{j\in\mathbf{P}\setminus\{i\}} (\alpha_{i,j} + \beta_{j,i}))$.

The signing algorithm outputs the round-2 protocol message $\mathsf{pm}_i^{(2)} := (w_i, u_i)$ of party i. It is assumed that the presigning secret state pst_i is used only once and is erased after use.

Combine: On receiving $\{(\mathsf{pm}_i^{(1)}, \mathsf{pm}_i^{(2)})\}_{i\in\mathbf{P}}$, the combining algorithm computes r as specified above, and sets $w := \sum_{i\in\mathbf{P}} w_i$ and $u := \sum_{i\in\mathbf{P}} u_i$. It then computes $\sigma := w \cdot u^{-1} \bmod q$. It outputs the signature (r, σ) if $\mathsf{ECDSA.Verify}(X, \mathsf{msg}, (r, \sigma)) = 1$, and $\perp$ otherwise.

Correctness. We briefly outline the correctness of the protocol. By the correctness of NIM, for each pair of signers $i, j \in \mathbf{P}, i \neq j$, we have $\alpha_{i,j} + \beta_{i,j} = k_i\gamma_j$, and $\mu_{i,j} + \nu_{i,j} = (\lambda_{i,\mathbf{P}} x_i)\gamma_j$. Define $k := \sum_{i\in\mathbf{P}} k_i$, $\gamma := \sum_{i\in\mathbf{P}} \gamma_i$, and $x := \sum_{i\in\mathbf{P}} \lambda_{i,\mathbf{P}} x_i$. Rearranging the combining step, we have $w = \gamma \cdot (m + rx)$ and $u = \gamma \cdot (zk + y)$. Note that $R = g^{zk+y}$ and $r = R|_{x\text{-axis}}$. Therefore, we have $\sigma = w/u = (zk + y)^{-1}(m + rx)$, and we can conclude that (r, σ) is a valid signature on msg under verification key X.

3.2 Security Analysis

Below, we prove that our two-round threshold ECDSA scheme satisfies existential unforgeability under chosen message attack with static corruption (definition 3) using the building blocks in the random oracle model. The theorem is as follows:

Theorem 1. *Assuming the Doubly-Enhanced Unforgeability of ECDSA and the security of NIM and NIZKAoK schemes, with hash functions* H_1, H_2 *modeled as random oracles, the above threshold ECDSA protocol satisfies static unforgeability.*

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ that wins the static unforgeability experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{TSUF}}$ w.r.t. our threshold ECDSA protocol with non-negligible probability. We show that, under the given conditions, there exists a PPT algorithm $\mathcal{B}$ that can win the experiment of Doubly-Enhanced Unforgeability of ECDSA $\mathsf{Exp}_{\mathcal{B}}^{\mathsf{DEUF}}$ with non-negligible probability. Below, we describe how the reduction $\mathcal{B}$ proceeds.

Initialization. The reduction $\mathcal{B}$ initializes the experiment $\mathsf{Exp}_{\mathcal{B}}^{\mathsf{DEUF}}$, and receives the public parameters pp from the challenger. It then sets up the public parameters for the threshold ECDSA protocol using what's received. Moreover, $\mathcal{B}$ will act as random oracles for hash queries to H_1 and H_2.

The reduction $\mathcal{B}$ receives the verification key X from $\mathsf{Exp}_{\mathcal{B}}^{\mathsf{DEUF}}$. To win the experiment, it will have to forge a signature on an unsigned message under X. Towards this goal, $\mathcal{B}$ first embeds the verification key X as the one output by KeyGen. This can be done in a way that is standard in threshold cryptography. Since $\mathcal{B}$ does not know the discrete logarithm of X, it gives each corrupted party $j \in \mathbf{C}$ a secret-key share $x_i \leftarrow\$ \mathbb{Z}_q$ that is random. Their public shares $\{X_j\}_{j\in\mathbf{C}}$ are each computed as $X_j = g^{x_j}$. Then, $\mathcal{B}$ simulates consistent public shares $\{X_i\}_{i\in\mathbf{H}}$ for the honest parties without knowing the corresponding discrete logarithms, using Lagrange interpolation in the exponent.

We remark that the simulation of KeyGen is also possible when a DKG subprotocol is used, on condition that it is simulatable given X. In such a case, the reduction $\mathcal{B}$ extracts the secret-key shares $\{x_j\}_{j\in\mathbf{C}}$ of corrupted parties using the NIZKAoKs $\{\pi_{x,j}\}_{j\in\mathbf{C}}$.

The reduction $\mathcal{B}$ then proceeds to simulate the presigning and signing interactions with $\mathcal{A}$. It plays the roles of the honest parties $i \in \mathbf{H}$ and answers the queries to oracles Presign and Sign made by the adversary $\mathcal{A}$. It manages query sets as described in $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{TSUF}}$, and additionally initializes another one Q_{presign} as empty.

Simulating Presigning. On receiving $\mathcal{A}$'s query (i, sid) to oracle Presign, if the query is invalid, i.e., it does not pass the checks in $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{TSUF}}$, then the simulated oracle Presign returns $\perp$. Otherwise, the query for a presigning protocol message is valid, and $\mathcal{B}$ proceeds as follows.

First, $\mathcal{B}$ checks if no item $(\mathsf{sid}, \cdot)$ exists in Q_{presign}, i.e., sid is queried for the first time. If so, $\mathcal{B}$ queries oracle Presign in $\mathsf{Exp}_{\mathcal{B}}^{\mathsf{DEUF}}$ and receives a presignature R, and appends (sid, R) to Q_{presign}. If (sid, R) already exists, $\mathcal{B}$ fetches R previously recorded.

In either case, $\mathcal{B}$ samples a random $c_i \leftarrow\$ \mathbb{Z}_q$, and sets $K_i := R^{c_i}$. It then chooses γ_i randomly from $\mathbb{Z}_q$ and proceeds honestly in computing Γ_i and $\mathsf{pe}_{\gamma,i}$. Since $\mathcal{B}$ does not know the discrete logarithm of K_i, it samples a random $k_i \leftarrow\$ \mathbb{Z}_q$, computes the NIM encoding $\mathsf{pe}_{k,i}$ with input k_i, and simulates the NIZKAoK π_i using the zero-knowledge simulator. Finally, $\mathcal{B}$ outputs party i's presigning protocol message $\mathsf{pm}_i^{(1)} := (K_i, \Gamma_i, \mathsf{pe}_{k,i}, \mathsf{pe}_{\gamma,i}, \pi_i)$ to $\mathcal{A}$.

Simulating Hash Queries. The reduction $\mathcal{B}$ simulates queries to H_1 using standard lazy sampling. It maintains a list of queries and responses, and samples a random $z \leftarrow\$ \mathbb{Z}_q$ if the query is new. If the query is repeated, $\mathcal{B}$ returns the same response as before. However, when $\mathcal{A}$ queries H_1 on a new input $(X, \mathsf{msg}, \{\mathsf{pm}_j^{(1)}\}_{j\in\mathbf{P}})$, before returning $z \leftarrow\$ \mathbb{Z}_q$ to $\mathcal{A}$, the reduction $\mathcal{B}$ programs H_2 on input z in a special way. Namely, for each corrupted signer $j \in \mathbf{P} \cap \mathbf{C}$, the reduction $\mathcal{B}$ parses $\mathsf{pm}_j^{(1)}$ as $(K_j, \Gamma_j, \mathsf{pe}_{k,j}, \mathsf{pe}_{\gamma,j}, \pi_j)$, and if all NIZKAoKs $\{\pi_j\}_{j\in\mathbf{P}\cap\mathbf{C}}$ pass verification, $\mathcal{B}$ does the following:

- Extract $\{(k_j, \gamma_j)\}_{j\in\mathbf{P}\cap\mathbf{C}}$ using the NIZKAoK extractor;
- Query oracle Tweak in $\mathsf{Exp}_{\mathcal{B}}^{\mathsf{DEUF}}$ with input (R, z', m), where $z' := z \cdot \sum_{j\in\mathbf{P}\cap\mathbf{H}} c_j$ and $m := \mathsf{H}_{\mathsf{sig}}(\mathsf{msg})$, receiving μ in return;
- Program H_2 to return $y := \mu - z \cdot \sum_{j\in\mathbf{P}\cap\mathbf{C}} k_j$ on input z.

If any NIZKAoK fails verification, then $\mathcal{B}$ programs H_2 to return a random $y \leftarrow\$ \mathbb{Z}_q$ on input z. If a new query to H_2 on a previously unknown input is made, then $\mathcal{B}$ also samples a random $y \leftarrow\$ \mathbb{Z}_q$ and returns it. We provide some intuition here. Denote $K = \prod_{i\in\mathbf{P}} K_i$ as prescribed. In the online phase corresponding to the queried transcript, the presignature will be re-randomized from K to

$$K^z g^y = \left(\prod_{i\in\mathbf{P}} K_i\right)^z g^y = \left(\prod_{i\in\mathbf{P}\cap\mathbf{H}} R^{c_i} \cdot \prod_{j\in\mathbf{P}\cap\mathbf{C}} g^{k_j}\right)^z \cdot g^{\mu - z\cdot\sum_{j\in\mathbf{P}\cap\mathbf{C}} k_j} = R^{z'} g^{\mu}$$

Since R is given by the $\mathsf{Exp}^{\mathsf{DEUF}}_{\mathcal{B}}$ challenger, z' is the tweak induced by $\mathcal{B}$, and μ is the re-randomizer induced by the $\mathsf{Exp}^{\mathsf{DEUF}}_{\mathcal{B}}$ challenger, this embeds a presignature, which the $\mathsf{Exp}^{\mathsf{DEUF}}_{\mathcal{B}}$ challenger can use for single-party signing, into a simulated possibly-correct threshold signing session.

Simulating Signing. On receiving $\mathcal{A}$'s query to oracle Sign with input $(i, \mathsf{sid}, \mathbf{P}, \mathsf{msg}, \{\mathsf{pm}^{(1)}_j\}_{j\in\mathbf{P}})$, if the query fails any check defined in $\mathsf{Exp}^{\mathsf{TSUF}}_{\mathcal{A}}$, or if any NIZKAoK from a corrupted signer $j \in \mathbf{P}\cap\mathbf{C}$ fails verification, then the simulated oracle Sign returns $\perp$. Otherwise, the query for an online signing protocol message is valid, and $\mathcal{B}$ proceeds as follows.

First, $\mathcal{B}$ queries $\mathsf{H}_1(X, \mathsf{msg}, \{\mathsf{pm}^{(1)}_j\}_{j\in\mathbf{P}})$, so that m, z, z', μ, y are well-defined and $\{(k_j, \gamma_j)\}_{j\in\mathbf{P}\cap\mathbf{C}}$ are extracted, if not already.

If $(\mathsf{sid}, \mathsf{msg})$ is new and does not already exist in $\mathcal{Q}_{\mathsf{sign}}$, then $\mathcal{B}$ does the following (note that this will define $\hat{k}_i, \hat{x}_i$ for every honest party $i \in \mathbf{P}\cap\mathbf{H}$ under the same sid):

- Query oracle Sign in $\mathsf{Exp}^{\mathsf{DEUF}}_{\mathcal{B}}$ with input (R, z', m, μ) and obtain σ, which is the right part of a valid signature on msg under X.
- Sample $u' \leftarrow\!\!\$\ \mathbb{Z}_q$ and set $w' := \sigma \cdot u'$.
- Define $\gamma := \sum_{j\in\mathbf{P}} \gamma_j$; this is feasible as $\{\gamma_j\}_{j\in\mathbf{P}\cap\mathbf{H}}$ are known by $\mathcal{B}$ and $\{\gamma_j\}_{j\in\mathbf{P}\cap\mathbf{C}}$ are extracted.
- Define $k' := u'(z\gamma)^{-1} - yz^{-1}$ such that $\gamma \cdot (zk' + y) = u'$.
- Sample $\hat{k}_j$ at random for each honest signer $j \in \mathbf{P}\cap\mathbf{H}$ such that $\sum_{j\in\mathbf{P}\cap\mathbf{H}} \hat{k}_j + \sum_{j\in\mathbf{P}\cap\mathbf{C}} k_j = k'$; this is feasible as $\{k_j\}_{j\in\mathbf{P}\cap\mathbf{C}}$ are extracted.
- Define $x' := w'(r\gamma)^{-1} - mr^{-1}$ such that $\gamma \cdot (m + rx') = w'$.
- Sample $\hat{x}_j$ at random for each honest signer $j \in \mathbf{P}\cap\mathbf{H}$ such that $\sum_{j\in\mathbf{P}\cap\mathbf{H}} \hat{x}_j + \sum_{j\in\mathbf{P}\cap\mathbf{C}} \lambda_{j,\mathbf{P}} x_j = x'$; this is feasible as $\{x_j\}_{j\in\mathbf{P}\cap\mathbf{C}}$ are known or extracted.

Then, $\mathcal{B}$ does the calculations in steps (1) to (4) of the online-phase protocol on behalf of party i, obtaining $\alpha_{i,j}, \beta_{j,i}, \mu_{i,j}, \nu_{j,i}$ and r. The simulated oracle Sign output to $\mathcal{A}$ is $(\hat{w}_i, \hat{u}_i)$, where

$$\hat{w}_i := m\gamma_i + r(\hat{x}_i\gamma_i + \textstyle\sum_{j\in\mathbf{P}\setminus\{i\}} (\mu_{i,j} + \nu_{j,i} - \lambda_{i,\mathbf{P}} x_i \gamma_j + \hat{x}_i \gamma_j)),$$

$$\hat{u}_i := y\gamma_i + z(\hat{k}_i\gamma_i + \textstyle\sum_{j\in\mathbf{P}\setminus\{i\}} (\alpha_{i,j} + \beta_{j,i} - k_i\gamma_j + \hat{k}_i\gamma_j)).$$

This embeds σ, the right part of a valid signature, into a simulated possibly-correct signing session.

Winning the Experiment. If $\mathcal{A}$ wins the simulated experiment, then by definition, it outputs a valid forgery $\mathsf{sig}^* = (r^*, \sigma^*)$ on a message msg^* that has not been queried to oracle Sign in $\mathsf{Exp}^{\mathsf{TSUF}}_{\mathcal{A}}$. Then, $\mathcal{B}$ can forward $(\mathsf{msg}^*, \mathsf{sig}^*)$ to the challenger of $\mathsf{Exp}^{\mathsf{DEUF}}_{\mathcal{B}}$ and win it, thus completing the reduction.

Analysis of Reduction. To complete the proof, we must argue that the reduction $\mathcal{B}$ only fails with negligible probability, and that the simulation is indistinguishable from the real threshold signature static unforgeability experiment $\mathsf{Exp}^{\mathsf{TSUF}}_{\mathcal{A}}$ from the view of $\mathcal{A}$.

Claim 1. *The reduction $\mathcal{B}$ only fails with negligible probability.*

We consider the following bad events that will force $\mathcal{B}$ to abort.

- *Hash collisions.* If any hash query results in a collision, $\mathcal{B}$ aborts. By the birthday bound, this happens with negligible probability for a polynomial number of queries.
- *NIZKAoK extraction failure.* If the NIZKAoK extraction fails, $\mathcal{B}$ aborts. As we assume knowledge extractability of NIZKAoKs, this happens with negligible probability.
- H_2 *programming conflicts.* The reduction $\mathcal{B}$ aborts if it fails to program H_2 on a random input z in the specified way because a query of $\mathsf{H}_2(z)$ was already made. This happens if $\mathcal{A}$ has guessed $\mathcal{B}$'s random choice of z correctly, with negligible probability.

Claim 2. *For the adversary $\mathcal{A}$, the interaction with $\mathcal{B}$ is indistinguishable from the real experiment $\mathsf{Exp}^{\mathsf{TSUF}}_{\mathcal{A}}$.*

In the KeyGen phase, since $\mathcal{B}$ does not know the discrete logarithms of simulated $\{X_i\}_{i\in\mathbf{H}}$, it simulates NIM public encodings $\{\mathsf{pe}_{x,i}\}_{i\in\mathbf{H}}$ using random inputs. Also, when simulating oracle Presign, since $\mathcal{B}$ does not know the discrete logarithms of simulated $\{K_i = R^{c_i}\}_{i\in\mathbf{P}\cap\mathbf{H}}$, it simulates NIM public encodings $\{\mathsf{pe}_{k,i}\}_{i\in\mathbf{P}\cap\mathbf{H}}$ using random inputs. The NIZKAoKs are also simulated, but since the NIZKAoK scheme is zero-knowledge, this will not be detected.

Indistinguishability of simulated oracle Presign can be expressed as the computational indistinguishability of distributions

$$\left\{(g^{k_i}, \mathsf{pe}_{k,i}) \mid k_i \leftarrow\!\!\$\ \mathbb{Z}_q;\ (\mathsf{pe}_{k,i}, \mathsf{st}_{k,i}) \leftarrow \mathsf{NIM.Encode}_B(\mathsf{crs}, k_i)\right\}$$
$$\overset{c}{\approx} \left\{(R^{c_i}, \mathsf{pe}_{k,i}) \;\middle|\; \begin{array}{l} R \leftarrow\!\!\$\ \mathbb{G};\ c_i \leftarrow\!\!\$\ \mathbb{Z}_q;\ k_i \leftarrow\!\!\$\ \mathbb{Z}_q; \\ (\mathsf{pe}_{k,i}, \mathsf{st}_{k,i}) \leftarrow \mathsf{NIM.Encode}_B(\mathsf{crs}, k_i) \end{array}\right\}.$$

By a purely syntactic change, this is equivalent to

$$\left\{(g^x, \mathsf{pe}_x) \mid x \leftarrow\!\!\$\ \mathbb{Z}_q;\ (\mathsf{pe}_x, \mathsf{st}) \leftarrow \mathsf{NIM.Encode}_B(\mathsf{crs}, x)\right\}$$
$$\overset{c}{\approx} \left\{(g^x, \mathsf{pe}_y) \mid x, y \leftarrow\!\!\$\ \mathbb{Z}_q;\ (\mathsf{pe}_y, \mathsf{st}) \leftarrow \mathsf{NIM.Encode}_B(\mathsf{crs}, y)\right\},$$

which also expresses indistinguishability in the KeyGen phase. This follows from the input privacy of NIM. A reduction proceeds simply as follows: it samples x, y and obtains pe, an NIM public encoding of x or y, and then forwards (g^x, pe) to $\mathcal{A}$; If $\mathcal{A}$ can tell from which distribution it is drawn, then it can break the said property.

The simulated random oracles $\mathsf{H}_1, \mathsf{H}_2$ are indistinguishable from real. Each z is sampled uniformly at random by $\mathcal{B}$. Each y is also uniformly random: when it is derived as $y = \mu - \sum_{j\in\mathbf{P}\cap\mathbf{C}} k_j$, since μ is sampled uniformly at random by the $\mathsf{Exp}^{\mathsf{DEUF}}_{\mathcal{B}}$ challenger and hidden from $\mathcal{A}$, we know y is uniformly random. Finally, the simulated oracle Sign responses are statistically close to real. The only constraint on $\{w_i, u_i\}_{i\in\mathbf{P}}$ is that their sums w, u divide to the right part σ of the one valid signature on msg under the verification key X conditioned on presignature $K^z g^y$.

□

4 CONCRETE INSTANTIATION

In this section, we provide information on how we instantiate our building blocks, namely NIM and accompanying NIZKAoKs. We do not claim novelty for results presented here, and refer the reader to original sources for their proofs.

Justifying the Choice. Abram et al. [3] show that NIM can be instantiated (without a non-negligible correctness error) from Decisional Composite Residuosity (DCR) in the Paillier group $\mathbb{Z}^*_{N^2}$, Quadratic Residuosity (QR) in the RSA group $\mathbb{Z}_N^*$, a variant of Joye–Libert, DDH-like assumptions in class groups, or LWE with a superpolynomial modulus-to-noise ratio.

We opt for an instantiation using class groups due to the following considerations. First, because we work in a malicious setting, the mismatching message space of Paillier/RSA instantiations lead to security problems that must be addressed using range proofs, which are rather expensive. This is evidenced by prior threshold ECDSA designs that use Paillier encryption with range proofs [34, 15]. In contrast, with the parameters set properly, class-group encryption can natively handle a message space of a field of integers modulo prime q, thus removing the need of range proofs. Second, class-group elements are much smaller than RSA/Paillier ones at the same level of security, mainly because the hardness of class-group assumptions does not require the hardness of factoring. Moreover, optimized class-group operations are faster than those in Paillier groups; this is contrary to earlier widely-held beliefs based on less efficient implementations. For information on the efficiency of class-group cryptography, we refer the reader to Bouvier et al. [11].

4.1 Background on Class Groups

Group Structure. We work in a finite abelian group $\hat{G}$. Inside $\hat{G}$, a cyclic subgroup F is generated by $f \in \hat{G}$ of order q, and discrete logarithms base f can be very efficiently computed. One can also sample elements in $\hat{G}$ that generate 'hard' subgroups. Each hard subgroup is also cyclic, but it is hard to compute discrete logarithms in it or to find the order of it.

Distributed Dlog. Suppose two parties hold two correlated elements of the group $\hat{G}$, say Alice holds $f^z h$ and Bob holds h, where $z \in \mathbb{Z}_q$ and h is in a hard subgroup. There exists a Distributed Discrete Logarithm (DDLog) mechanism that allows them to obtain an additive sharing of z without any interaction; that is, Alice can obtain $z_A \in \mathbb{Z}_q$ and Bob can obtain $z_B \in \mathbb{Z}_q$ such that $z_A + z_B = z$.

The DDLog mechanism relies on a *coset labelling function* ϕ that, for each coset C of F, deterministically maps all elements in C to a specific one among them. More intuitively, for each h in a hard subgroup, let $C_h := \{f^x h \mid x \in \mathbb{Z}_q\}$, and there exists $\delta \in C_h$ such that for each $e \in C_h$ we have $\phi(e) = \delta$.

Continuing the description of DDLog: Alice computes $f^z h/\phi(f^z h)$, and Bob computes $\phi(h)/h$. It is straightforward to see that these results lie in the easy subgroup F. Calculating the discrete logarithms in F gives the two shares z_A and z_B, and they sum to z.

Under the Hood. The reader can find information on class-group cryptography in [11] and on the coset labelling function in [1]. Very roughly, the underlying set of $\hat{G}$ is squares in the class group of binary quadratic forms of discriminant $-pq^3$, where p is a random 1,571-bit prime at 128-bit security; the group operation is Gauss composition. The easy subgroup F is generated by the form $f = (q^2, q, \frac{1+pq}{4})$. A generator of a hard subgroup can be sampled as t^q, where t is randomly sampled in $\hat{G}$. All random coins used above can be made public to eliminate trapdoors, hence class groups are often said to have a transparent setup. The coset labelling function is a composition of algorithms 1 and 2 in [45].

Sampling Exponents. Let $H := \langle h \rangle$ be a hard subgroup of $\hat{G}$, where h is a generator sampled using the approach above. In cryptographic applications, we need a distribution $\mathcal{D}_q$ over integers such that $\{h^x \mid x \in \mathcal{D}_q\}$ is statistically close to the uniform distribution over the hard subgroup H. There are several ways to define $\mathcal{D}_q$ [20, lemma 4]. Typically, it is the uniform distribution over $[2^{\kappa_{\mathrm{st}}}\tilde{s}]$, where κ_{st} is the statistical security parameter and $\tilde{s}$ is an upper bound on the order of the hard subgroup $|H|$. In practice, κ_{st} is set to 40 and $\tilde{s}$ is set to $2^{\lceil \log \sqrt{pq} \rceil}$.

4.2 NIM from Class Groups

The Non-Interactive Multiplication construction from class groups, roughly speaking, uses Pedersen commitment for Encode_A, and Castagnos–Laguillaumie (CL) encryption [19, 20] for Encode_B. In prior threshold ECDSA schemes building on CL encryption, the DDLog mechanism was not utilized, and multiplication proceeds as two-round OLE. However, it turns out that NIM from class groups is quite simple and does not add significant complexity to threshold ECDSA, either conceptually or computationally. More importantly, in our scheme it is not required that each party samples a key pair in the class group; two publicly known random generators g_0, g_1 of hard subgroups are sufficient. The construction is as follows.

NIM construction

- $\mathsf{Setup}(1^\kappa, q)$: Return $\mathsf{crs} := (p, q, f, g_0, g_1)$, where g_0, g_1 are two random generators of hard subgroups.
- $\mathsf{Encode}_A(\mathsf{crs}, v)$:
  (1) Sample $s \leftarrow\$ \mathcal{D}_q$;
  (2) Compute $\mathsf{pe}_A := g_0^s g_1^v$;
  (3) Set $\mathsf{st}_A := (s, v)$;
  (4) Return $(\mathsf{pe}_A, \mathsf{st}_A)$.
- $\mathsf{Encode}_B(\mathsf{crs}, v)$:
  (1) Sample $r \leftarrow\$ \mathcal{D}_q$;
  (2) Compute $\mathsf{pe}_B := (g_0^r, f^v g_1^r)$;
  (3) Set $\mathsf{st}_B := r$;
  (4) Return $(\mathsf{pe}_B, \mathsf{st}_B)$.
- $\mathsf{Decode}_A(\mathsf{crs}, \mathsf{pe}_B, \mathsf{st}_A)$:
  (1) Parse $(c_0, c_1) := \mathsf{pe}_B$, $(s, v) := \mathsf{st}_A$;
  (2) Compute $e := c_0^s c_1^v$;
  (3) Set $\delta := \phi(e)$;
  (4) Set $z_A := \mathsf{dlog}_f(e/\delta)$;
  (5) Return z_A.
- $\mathsf{Decode}_B(\mathsf{crs}, \mathsf{pe}_A, \mathsf{st}_B)$:
  (1) Parse $c := \mathsf{pe}_A$, $r := \mathsf{st}_B$;
  (2) Compute $e := c^r$;
  (3) Set $\delta := \phi(e)$;
  (4) Set $z_B := \mathsf{dlog}_f(\delta/e)$;
  (5) Return z_B.

Correctness. The correctness of the NIM scheme above directly follows from [3, theorem 5]. To build some intuition, consider the following informal explanation. Alice has input $x \in \mathbb{Z}_q$ and Bob has input $y \in \mathbb{Z}_q$. They interact in a simultaneous round, as follows:

(1) Alice sends $g_0^s g_1^x$ to Bob, and Bob sends $(g_0^r, f^y g_1^r)$ to Alice, where s, r are random numbers.

(2) Alice computes $(g_0^r)^s(f^y g_1^r)^x = f^{xy} g_0^{sr} g_1^{xr}$, and Bob computes $(g_0^s g_1^x)^r = g_0^{sr} g_1^{xr}$, after receiving each other's message.
(3) They compute a common label $\delta := \phi(f^{xy} g_0^{sr} g_1^{xr}) = \phi(g_0^{sr} g_1^{xr})$, where ϕ is the coset labelling function.
(4) Alice gets an additive share $z_A := \mathsf{dlog}_f(f^{xy} g_0^{sr} g_1^{xr}/\delta)$; Bob gets $z_B := \mathsf{dlog}_f(\delta/g_0^{sr} g_1^{xr})$. At this point it is easy to check that $z_A + z_B = xy$.

Input Privacy. The Encode_A algorithm hides the input under the uniformity assumption [3, definition 16]. The Encode_B algorithm hides the input under the Hard Subgroup Membership assumption [11, definition 2].

Definition 7 (Uniformity Assumption [3]). *Let $\hat{G}$ be the class group, and let $t \leftarrow\!\$\ \hat{G}$, $h := t^q$ such that h is a hard-subgroup generator. The Uniformity assumption holds if, for all PPT $\mathcal{A}$,*

$$\left|\Pr\left[\mathcal{A}(c) = 1 \,\middle|\, r \leftarrow\!\$\ \mathcal{D}_q, c := h^r\right] - \Pr\left[\mathcal{A}(c) = 1 \,\middle|\, c \leftarrow\!\$\ \hat{G}\right]\right| \le \mathsf{negl}(\kappa).$$

Definition 8 (Hard Subgroup Membership Assumption [11]). *Let $\hat{G}$ be the class group, and let $t \leftarrow\!\$\ \hat{G}$, $h := t^q$ such that h is a hard-subgroup generator. The Hard Subgroup Membership assumption holds if, for all PPT $\mathcal{A}$,*

$$\Big|\Pr\left[\mathcal{A}(c) = 1 \,\middle|\, r \leftarrow\!\$\ \mathcal{D}_q, c := h^r\right] - \Pr\left[\mathcal{A}(c) = 1 \,\middle|\, r \leftarrow\!\$\ \mathcal{D}_q, m \leftarrow\!\$\ \mathbb{Z}_q, c := f^m h^r\right]\Big| \le \mathsf{negl}(\kappa).$$

Theorem 2 (implicit in [3]). *If the Uniformity and Hard Subgroup Membership assumptions hold in the class group $\hat{G}$, then the NIM scheme above satisfies Input Privacy (definition 4).*

4.3 Concrete NIZKAoKs

Alongside the NIM instantiation above, we need NIZKAoKs for the following relations:

$$\mathcal{R}_{\text{CL-DL}} = \left\{(c_0, c_1, V); (r, v) \mid c_0 = g_0^r \wedge c_1 = f^v g_1^r \wedge V = g^v\right\};$$
$$\mathcal{R}_{\text{Ped-DL}} = \left\{(c, V); (r, v) \mid c = g_0^r g_1^v \wedge V = g^v\right\}.$$

In class groups, the knowledge extractability of Sigma protocols is a complicated matter. Using binary challenges results in special-sound proofs [18], but the prover must be repeated for many times, which incurs much overhead. Assuming *Low Order* (it is hard to find low-order elements in $\hat{G}$) and *Strong Root* (it is hard to find roots in $\hat{G}$ of random elements in a hard subgroup H), the Sigma-protocol proofs in [16] are shown to be knowledge-extractable unless some bad event happens which breaks either assumption. For these assumptions to hold, the hard-subgroup generators must not be adversarially chosen, and this is fine in our case. Further, it turns out that we do not really need to extract exponents in hard subgroups for our security proof of the threshold ECDSA protocol to go through, as is also observed in [14, 13, 5]. Under another *Rough Order* assumption (it is hard to distinguish between a random class group and one with rough order), general Sigma protocols can be proven to satisfy standard soundness, i.e., any adversary cannot produce a proof for a statement not in the language unless with negligible probability. This is enough for the detection of party misbehavior in the presigning phase.

In the security proof for threshold ECDSA, we require the secrets x_i, k_i, γ_i held by corrupted parties to be extracted from NIZKAoKs. As we use standard EQ composition of Sigma protocols to bind exponents in the class group $\hat{G}$ to ones in the elliptic curve group $\mathbb{G}$, the extractability of Sigma-protocol proofs in $\mathbb{G}$ suffices here.

The remaining question is how the reduction can extract witnesses from NIZKAoKs. Rewinding the adversary would blow up the running time of the reduction, as multiple corrupted parties are involved. On the other hand, if we aim for straight-line extractability from the Fischlin transform [31], the protocol would be much more expensive. In elliptic curve groups, a 10x increase of computation to generate a Fischlin proof may be acceptable, since it still takes less than 1 ms; however, each long exponentiation in the class group takes more than 5 ms at 128-bit security. Therefore, we adopt the approach of [4] to establish straight-line extractability for Schnorr-like Fiat-Shamir proofs in $\mathbb{G}$, within an idealized model such as the GGM.

NIZKAoK construction

Let $\mathsf{H}_{\mathsf{FS}} : \{0,1\}^* \to \mathbb{Z}_q$ be a hash function.

$\mathsf{NIZKAoK}_{\text{CL-DL}}$:

- $\mathsf{Prove}((c_0, c_1, V), (r, v))$:
  (1) Sample $\tilde{r} \leftarrow\!\$\ [2^{2\cdot\kappa_{\text{st}}} q\tilde{s}]$, $\tilde{v} \leftarrow\!\$\ \mathbb{Z}_q$.
  (2) Compute $\tilde{c_0} := g_0^{\tilde{r}}$, $\tilde{c_1} := f^{\tilde{v}} g_1^{\tilde{r}}$, $\tilde{V} := g^{\tilde{v}}$.
  (3) Compute $e := \mathsf{H}_{\mathsf{FS}}(g_0, g_1, f, g, c_0, c_1, V, \tilde{c_0}, \tilde{c_1}, \tilde{V})$.
  (4) Compute $s_r := \tilde{r} + e \cdot r \in \mathbb{Z}$, $s_v := \tilde{v} + e \cdot v \bmod q$.
  (5) Return $\pi := (\tilde{c_0}, \tilde{c_1}, \tilde{V}, s_r, s_v)$.
- $\mathsf{Verify}((c_0, c_1, V), \pi)$:
  (1) Parse $(\tilde{c_0}, \tilde{c_1}, \tilde{V}, s_r, s_v) := \pi$.
  (2) Compute $e := \mathsf{H}_{\mathsf{FS}}(g_0, g_1, f, g, c_0, c_1, V, \tilde{c_0}, \tilde{c_1}, \tilde{V})$.
  (3) Check that $s_r < 2^{2\cdot\kappa_{\text{st}}} q\tilde{s}$ and $s_v \in \mathbb{Z}_q$.
  (4) Check $g_0^{s_r} = \tilde{c_0} \cdot c_0^e$, $f^{s_v} g_1^{s_r} = \tilde{c_1} \cdot c_1^e$, $g^{s_v} = \tilde{V} \cdot V^e$.
  (5) If all checks pass, output 1; otherwise, output 0.

$\mathsf{NIZKAoK}_{\text{Ped-DL}}$:

- $\mathsf{Prove}((c, V), (r, v))$:
  (1) Sample $\tilde{r} \leftarrow\!\$\ [2^{2\cdot\kappa_{\text{st}}} q\tilde{s}]$, $\tilde{v} \leftarrow\!\$\ [2^{\kappa_{\text{st}}} q^2]$.
  (2) Compute $\tilde{c} := g_0^{\tilde{r}} g_1^{\tilde{v}}$, $\tilde{V} := g^{\tilde{v}}$.
  (3) Compute $e := \mathsf{H}_{\mathsf{FS}}(g_0, g_1, g, c, V, \tilde{c}, \tilde{V})$.
  (4) Compute $s_r := \tilde{r} + e \cdot r \in \mathbb{Z}$, $s_v := \tilde{v} + e \cdot v \in \mathbb{Z}$.
  (5) Return $\pi := (\tilde{c}, \tilde{V}, s_r, s_v)$.
- $\mathsf{Verify}((c, V), \pi)$:
  (1) Parse $(\tilde{c}, \tilde{V}, s_r, s_v) := \pi$.
  (2) Compute $e := \mathsf{H}_{\mathsf{FS}}(g_0, g_1, g, c, V, \tilde{c}, \tilde{V})$.
  (3) Check that $s_r < 2^{2\cdot\kappa_{\text{st}}} q\tilde{s}$ and $s_v < 2^{\kappa_{\text{st}}} q^2$.
  (4) Check that $g_0^{s_r} g_1^{s_v} = \tilde{c} \cdot c^e$ and $g^{s_v} = \tilde{V} \cdot V^e$.
  (5) If all checks pass, output 1; otherwise, output 0.

Definition 9 (Low Order Assumption [16]). *Let $\hat{G}$ be the class group. The Low Order assumption holds if, for all PPT $\mathcal{A}$,*

$$\Pr\left[\mu^d = 1 \in \hat{G} \wedge \mu \ne 1 \wedge |d| < 2^\kappa \mid (\mu, d) \leftarrow \mathcal{A}(\hat{G})\right] \le \mathsf{negl}(\kappa).$$

Definition 10 (Strong Root Assumption [16]). *Let H be a hard subgroup of the class group $\hat{G}$. The Strong Root assumption holds if, for all PPT $\mathcal{A}$,*

$$\Pr\left[X^e = Y \wedge \nexists k \in \mathbb{Z} \text{ s.t. } e = 2^k \mid Y \leftarrow\!\$\ H, (X, e) \leftarrow \mathcal{A}(Y)\right] \le \mathsf{negl}(\kappa).$$

THEOREM 3 ([16, 39]). *Assuming Low Order and Strong Root in $\hat{G}$, with H_{FS} modeled as a random oracle, the NIZKAoK scheme above is complete, zero-knowledge, and knowledge-extractable.*

5 IMPLEMENTATION AND EVALUATION

We present performance metrics of our threshold ECDSA protocol, and compare it with prior work. We implement our protocol based on the BICYCL library [11] which provides class group operations. In line with other open-source threshold ECDSA implementations, we target 128-bit security, which covers usage with the Bitcoin curve secp256k1. The code is available at https://anonymous.4open.science/r/tecdsa-2r-rust-8C3B.

5.1 Bandwidth Efficiency

We analyze the bandwidth usage in the signing protocol. Recall that the elliptic curve group is denoted $\mathbb{G}$ and the class group is denoted $\hat{G}$. In the presigning phase, each party sends two $\mathbb{G}$ elements and three $\hat{G}$ elements, excluding the NIZKAoKs. In the online signing phase, each party sends two $\mathbb{Z}_q$ elements. At 128-bit computational security and 40-bit statistical security, each $\hat{G}$ element takes 220 bytes, each $\mathbb{G}$ element takes 33 bytes, and each $\mathbb{Z}_q$ element takes 32 bytes. Therefore, communication excluding the proof is 790 bytes.

Flexible NIZKAoK Bandwidth. The size of the NIZKAoK sent in the presigning phase is calculated separately, because several variants may be considered in different practical settings. With the default formulation, each $\mathsf{NIZKAoK}_{\mathsf{CL\text{-}DL}}$ proof consists of two $\hat{G}$ elements, a $\mathbb{G}$ element, a $\mathbb{Z}_q$ element, and an integer that is 157 bytes long; each $\mathsf{NIZKAoK}_{\mathsf{Ped\text{-}DL}}$ proof consists of a $\hat{G}$ element, a $\mathbb{G}$ element, a 157-byte integer, and a 69-byte integer. Hence, the default size of the proof in the presigning phase is 1,141 bytes.

One can utilize the Schnorr signature size reduction technique to significantly reduce the proof size. Namely, for a Sigma protocol transcript (A, e, s), where e is derived from a hash function via the Fiat-Shamir heuristic, one can take (e, s) instead of (A, s) as the NIZKAoK proof. To verify such a proof, the first-round commitment A is reverse-sampled according to the verification equation, and if the Fiat-Shamir hash digest equals e, then the proof is accepted. Note that reverse sampling the first-round commitment is fast in both $\mathbb{G}$ and $\hat{G}$, and therefore the computation to verify such a proof is equal to that in the default formulation done naively. However, in this case one cannot accelerate it via batch verification. This does not cause a problem for not-too-many signers. In this setting the proof takes only 447 bytes.

If presigning is performed in a known batch size, then one can also utilize batch proofs [5]. In this setting, one proof can be used to verify the well-formedness of m presigning protocol messages with a size that only increases logarithmically with m. The verification computation, which dominates overall computation, cannot be reduced if done naively, but like in the default formulation, some precomputation can result in a decent speedup.

We report the most conservative number, $790 + 1141 = 1931$ bytes, as the total message length each party sends in each signing session, and do not delve into further details.

Comparison. If protocol messages are relayed by centralized servers, the total outbound communication of a party in our signing protocol is 1.9 KiB. This model is seen in threshold Schnorr schemes, e.g. [6], and we believe it accounts for many practical deployments. In this model, among other threshold ECDSA schemes, only [49] similarly achieves a constant communication of 3.4 KiB. Others all take linear communication per party due to pairwise OLE.

However, among prior threshold ECDSA papers, the common choice has been to measure communication in the peer-to-peer network model. In this case, all schemes require outbound communication per party that is at least linear in the number of parties. For our protocol, the number is $1.9 \cdot (t - 1)$ KiB.

We compare each party's outbound communication with each peer among different threshold ECDSA schemes in table 2, following [15, figure 1]. Note that [15] gives estimates using 112-bit security for Paillier encryption (2048-bit modulus N). Since 128-bit security (3072-bit modulus) is practically used in well-known open-source implementations,[2] we also list numbers at 128-bit security accordingly. For schemes that use broadcasts, our estimates take broadcasts as multicasts and do not count the overhead, which favors those schemes.

It is generally agreed that schemes based on OT consume more bandwidth than those based on homomorphic encryption, and schemes based on Paillier consume more bandwidth than those based on Castagnos–Laguillaumie class group encryption. Our protocol instantiated with class groups can be seen as one that extends the last technical approach by getting rid of the OLE response messages. In the peer-to-peer model, our scheme approximately

Table 2: Outbound communication with each peer among threshold ECDSA schemes at 128-bit security.

Scheme	Tool	Comm.	Note
GG18 [34]	Paillier enc	11.7 KiB	$\kappa = 112$
		16.7 KiB	$\kappa = 128$
LN18 [43]	Paillier enc	7.8 KiB	$\kappa = 112$
		10.8 KiB	$\kappa = 128$
	OT	190 KiB	original ver.
		60 KiB	updated ver.
CGGMP20 [15]	Paillier enc	16 KiB	4-round, $\kappa = 112$
		23 KiB	4-round, $\kappa = 128$
		15.8 KiB	7-round, $\kappa = 112$
		22.8 KiB	7-round, $\kappa = 128$
DKLs19 [29]	OT	90 KiB	
DKLs24 [30]	OT	50 KiB	
CCLST20 [16]	CL enc	3.5 KiB	
CCLST23 [17]	CL enc	4.2 KiB	
WMC24 [49]	threshold CL enc	3.4 KiB	
ANOSS22 [2]	ring-LPN PCG	$0.017n + 0.18$ KiB *	
This work	class group NIM	1.9 KiB	

*[2] reports comm. amortized over a batch of 94019 signatures; their scheme only supports full threshold ($t = n$) and is not threshold-optimal. Some listed schemes require broadcasts, the overhead of which is not counted.

[2] For example, the implementation of Dfns and the Linux Foundation Decentralized Trust project, available at https://github.com/LFDT-Lockness/cggmp21.

Table 3: Single-threaded signer computation time in milliseconds among threshold ECDSA schemes at 128-bit security.

Scheme	3-party	5-party	7-party	Note
CGGMP20 [15]	918	1810	2720	four-round ver.
CCLST20 [16]	388	628	868	
DKLs24 [30]	21	41	61	
WMC24 [49]	281	411	535	
This work	224	405	586	

Each number is the combined running time of presigning and signing. Based on local benchmarking and not accounting for network latency.

halves communication compared to some most bandwidth-efficient schemes [16, 17]. If a relaying server is available, our bandwidth savings would even be demonstrated asymptotically. Furthermore, as is explained in [4], a total communication of just 1.9 KiB makes our scheme very suitable for scenarios with constrained communication such as NFC, QR codes, and push notifications.

The scheme of Abram et al. [2] is the only one that consumes less bandwidth than ours, based on reported communication amortized over 94019 signing sessions. Its limitations are heavier computation, unimplemented preprocessing protocol, and only supporting full threshold (i.e., $t = n$) and not being threshold-optimal.

5.2 Computational Efficiency

We measure computational cost in terms of single-threaded running time of each party in a signing session. The machine used for the benchmarks features Intel i7-13700K CPU and 32 GB of memory, and runs Ubuntu 24.04. Results are presented in table 3.

Our comparison is limited in scope because not many prior schemes have open-source reference implementations. Nonetheless, the schemes listed in table 3 are reasonably representative of the major technical approaches, namely Paillier, OT, and class groups. Furthermore, in practical cryptocurrency wallets, the number of parties are typically 3 to 5 and rarely exceeds 10, and we thus focus on such small-scale deployments in table 3. We remark that in decentralized finance applications there can sometimes be more than 20 parties; however, as the listed schemes all have linear computation, the performance can be extrapolated in this case.

The computation of schemes based on OT, e.g. DKLs24 [30], primarily consists of symmetric cryptographic operations (via OT extension techniques) and can benefit from existing highly-optimized implementations. Therefore, it is rather hard for schemes like ours to outperform them computationally. However, for [15, 30] and our scheme, the overwhelming majority of computation is in the presigning phase, and the online-signing round takes less than 1 millisecond of computation; hence, with presigning, the difference in computation would not be pronounced for end users. Even when the protocol falls back to interactive signing (without available presignatures) our scheme would be fast enough. Threshold ECDSA is often used in applications with a human factor, such as cryptocurrency wallets where the user must check the transaction before signing. The average human reaction time to a visual stimulus is around 250 milliseconds, and any latency below that is generally acceptable. Anyway, since ours requires one fewer round and much less communication, it will outperform theirs in certain scenarios.

Our scheme runs faster than CGGMP20 [15] by roughly 4x, mainly because [15] relies on computationally expensive range proofs, which other schemes based on Paillier encryption also do. Recent works [51, 48] report improvements to them by optimizing the range proofs; their reported speedups are less than 2x, and we project that ours would still outperform these improved variants. At higher security levels, the advantage of schemes based on class groups over Paillier ones will be greater; see [11, table 3].

Our scheme outperforms CCLST20 [16] by 30% to 40%, and is basically on par with WMC24 [49]. However, for WMC24, a significant proportion of computation happens in the online-signing round, because their last step is threshold decryption, which requires class group exponentiations linear in n. With similar total computational costs, it is certainly preferable to offload the bulk of the computation to the presigning phase, which ours does. Moreover, WMC24 takes 4 rounds, 3 of which are broadcasts. We have not implemented CCLST23 [17], which is the identifiable-abort version of CCLST20 [16], and WMYC23 [50], but [49] reports that both are at least 2x slower than WMC24.

The ANOSS22 [2] scheme requires a large batch of presigning material to be expanded from preshared seeds at once. They report that, at a batch size of 94019, each signature takes more than 1 second of computation, and the total time for the whole batch is quite high. Their scheme appears suitable for scenarios where computing power is ample but communication is extremely constrained.

Further Optimizations. Our implementation is at a prototyping stage, without many in-depth optimizations. However, there are clear-cut paths towards speedups. We discuss some of them here. In our scheme, the computation that increases linearly with the number of parties consists in NIM decoding and NIZKAoK verification. To optimize the latter, if presigning is done in large batches, one can consider batch proofs [5], and if there are many parties, one can consider randomized batch verification [7]. In both scenarios, precomputation is crucial for accelerating class group exponentiations. In [5], a speedup of 4.7x for fixed-basis class-group exponentiation at a cost of 1.2 seconds of precomputation is reported. Using this strategy, NIZKAoK verification can be much faster since it mainly requires fixed-basis exponentiations base g_0 and g_1.

6 CONCLUSION

In this work, we propose the first two-round threshold ECDSA scheme in the threshold-optimal setting, solving an important open question in an active area of research. We prove our scheme secure, and show its practical efficiency. It outperforms the state of the art in bandwidth efficiency. Its computational speed is also competitive, and we see promise for further optimizations.

Our protocol is proven secure under the static corruption model. For threshold Schnorr-like signatures, there have been advances in achieving security under adaptive corruption, but for threshold ECDSA, full adaptive security (without relying on the single-inconsistent-player model) remains an open challenge. It is possible to instantiate our template with other number-theoretic tools such as Paillier. However, it remains to find an alternative approach that also leads to a two-round solution and is both light in computation (comparably to OT extension) and compact in communication. We leave them as questions for future research.

ACKNOWLEDGEMENTS

This work is supported by National Key Research and Development Program of China (2022YFB2701700); National Natural Science Foundation of China (62472255, 62302271); Youth Innovation Team of Shandong Province (2024KJH79); Natural Science Foundation of Shandong Province, China (ZR2023MF045, ZR2023QF088); Natural Science Foundation of Qingdao, China (23-2-1-152-zyyd-jch).

Hong-Sheng Zhou was supported in part by NSF grant CNS-1801470 and a VCU Research Quest grant.

REFERENCES

[1] Damiano Abram, Ivan Damgård, Claudio Orlandi, and Peter Scholl. 2022. An algebraic framework for silent preprocessing with trustless setup and active security. In *CRYPTO 2022* (LNCS 13510). Springer, 421–452. DOI: 10.1007/978-3-031-15985-5_15.

[2] 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.

[3] Damiano Abram, Lawrence Roy, and Peter Scholl. 2024. Succinct homomorphic secret sharing. In *EUROCRYPT 2024* (LNCS 14656). Springer, 301–330. DOI: 10.1007/978-3-031-58751-1_11.

[4] 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).

[5] Agathe Beaugrand, Guilhem Castagnos, and Fabien Laguillaumie. 2025. Efficient succinct zero-knowledge arguments in the CL framework. *Journal of Cryptology*, 38, 1, 13. DOI: 10.1007/s00145-024-09534-1.

[6] Mihir Bellare, Elizabeth C. Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. 2022. Better than advertised security for non-interactive threshold signatures. In *CRYPTO 2022* (LNCS 13510). Springer, 517–550. DOI: 10.1007/978-3-031-15985-5_18.

[7] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. 2012. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2, 2, 77–89. DOI: 10.1007/s13389-012-0027-1.

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

[9] Dan Boneh, Iftach Haitner, and Yehuda Lindell. 2025. Exponent-VRFs and their applications. In *EUROCRYPT 2025*. Cryptology ePrint Archive, Paper 2024/397.

[10] 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.

[11] 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, 17. DOI: 10.1007/s00145-023-09459-1.

[12] Elette Boyle, Lalita Devadas, and Sacha Servan-Schreiber. 2025. Non-interactive distributed point functions. In *PKC 2025*. Cryptology ePrint Archive, Paper 2025/095.

[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] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] Megan Chen, Carmit Hazay, Yuval Ishai, Yuriy Kashnikov, Daniele Micciancio, Tarik Riviere, abhi shelat, Muthu Venkitasubramaniam, and Ruihan Wang. 2021. Diogenes: lightweight scalable RSA modulus generation with a dishonest majority. In *IEEE S&P 2021*, 590–607. DOI: 10.1109/SP40001.2021.00025.

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

[23] 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.

[24] Elizabeth Crites, Chelsea Komlo, and Mary Maller. 2023. Fully adaptive schnorr threshold signatures. In *CRYPTO 2023*. Springer, 678–709. DOI: 10.1007/978-3-031-38557-5_22.

[25] Anders P. K. Dalskov, Claudio Orlandi, Marcel Keller, Kris Shrishak, and Haya Shulman. 2020. Securing DNSSEC keys via threshold ECDSA from generic MPC. In *ESORICS 2020* (LNCS 12309). Springer, 654–673. DOI: 10.1007/978-3-030-59013-0_32.

[26] 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.

[27] 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.

[28] Yvo Desmedt and Yair Frankel. 1990. Threshold cryptosystems. In *CRYPTO '89* (LNCS 435). Springer, 307–315. DOI: 10.1007/0-387-34805-0_28.

[29] 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.

[30] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. 2024. Threshold ECDSA in three rounds. In *IEEE S&P 2024*. DOI: 10.1109/SP54263.2024.00178.

[31] 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.

[32] 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).

[33] Adam Gągol, Jędrzej Kula, Damian Straszak, and Michał Świętek. 2020. Threshold ECDSA for decentralized asset custody. Cryptology ePrint Archive, Paper 2020/498. (2020).

[34] 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.

[35] 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.

[36] Jens Groth and Victor Shoup. 2022. Design and analysis of a distributed ECDSA signing service. Cryptology ePrint Archive, Paper 2022/506. (2022).

[37] 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.

[38] Dominik Hartmann and Eike Kiltz. 2023. Limits in the provable security of ECDSA signatures. In *TCC 2023* (LNCS 14372). Springer, 279–309. DOI: 10.1007/978-3-031-48624-1_11.

[39] 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*, 4286–4300. DOI: 10.1145/3658644.3670312.

[40] Jonathan Katz and Antoine Urban. 2024. Honest-majority threshold ECDSA with batch generation of key-independent presignatures. Cryptology ePrint Archive, Paper 2024/2011. (2024).

[41] Chelsea Komlo and Ian Goldberg. 2020. FROST: flexible round-optimized schnorr threshold signatures. In *SAC 2020* (LNCS 12804). Springer, 34–65. DOI: 10.1007/978-3-030-81652-0_2.

[42] Yashvanth Kondi and Divya Ravi. 2025. Separating broadcast from cheater identification. Cryptology ePrint Archive, Paper 2025/052. (2025).

[43] 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.

[44] Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. 2021. The rise of paillier: homomorphic secret sharing and public-key silent OT. In *EUROCRYPT 2021* (LNCS 12696). Springer, 678–708. DOI: 10.1007/978-3-030-77870-5_24.

[45] Sachar Paulus and Tsuyoshi Takagi. 2000. A new public-key cryptosystem over a quadratic order with quadratic decryption time. *Journal of Cryptology*, 13, 2, 263–272. DOI: 10.1007/s001459910010.

[46] Michaella Pettit. 2021. Efficient threshold-optimal ECDSA. In *CANS 2021* (LNCS 13099). Springer, 116–135. DOI: 10.1007/978-3-030-92548-2_7.

[47] Victor Shoup. 1997. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT '97*. Springer, 256–266. DOI: 10.1007/3-540-69053-0_18.

[48] Guofeng Tang, Shuai Han, Li Lin, Changzheng Wei, and Ying Yan. 2024. Batch range proof: how to make threshold ECDSA more efficient. In *ACM CCS 2024*. ACM, 4256–4270. DOI: 10.1145/3658644.3670287.

[49] 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.

[50] 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.

[51] Zhikang Xie, Mengling Liu, Haiyang Xue, Man Ho Au, Robert H. Deng, and Siu-Ming Yiu. 2024. Direct range proofs for paillier cryptosystem and their applications. In *ACM CCS 2024*. ACM, 899–913. DOI: 10.1145/3658644.3690261.

[52] 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.

[53] 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.