Stateless Deterministic Multi-Party EdDSA Signatures with Low Communication

Qi Feng[1], Kang Yang[2,(✉)], Kaiyi Zhang[3], Xiao Wang[4], Yu Yu[3,6], and Xiang Xie[5,6]

[1] Wuhan University, fengqi.whu@whu.edu.cn
[2] State Key Laboratory of Cryptology, yangk@sklc.org
[3] Shanghai Jiao Tong University, kzoacn@sjtu.edu.cn, yuyu@yuyu.hk
[4] Northwestern University, wangxiao@northwestern.edu
[5] Primus Labs, xiexiangiscas@gmail.com
[6] Shanghai Qi Zhi Institute

Abstract. EdDSA is a standardized signing algorithm, by both the IRTF and NIST, that is widely used in blockchain, e.g., Hyperledger, Cardano, Zcash, etc. It is a variant of the well-known Schnorr signature scheme that leverages Edwards curves. It features stateless and deterministic nonce generation, meaning it does not rely on a reliable source of randomness or state continuity. Recently, NIST issued a call for multi-party threshold EdDSA signatures, with one approach verifying nonce generation through zero-knowledge (ZK) proofs.

In this paper, we propose a new stateless and deterministic multi-party EdDSA protocol in the full-threshold setting, capable of tolerating all-but-one malicious corruption. Compared to the state-of-the-art multi-party EdDSA protocol by Garillot et al. (Crypto'21), our protocol reduces communication cost by a factor of 56× while maintaining the same three-round structure, albeit with a roughly 2.25× increase in computational cost. We utilize information-theoretic message authentication codes (IT-MACs) in a multi-verifier setting to authenticate values and transform them from the Boolean domain to the arithmetic domain by refining multi-verifier extended doubly-authenticated bits (mv-edaBits). Additionally, we employ pseudorandom correlation functions (PCF) to generate IT-MACs in a stateless and deterministic manner. Combining these elements, we design a multi-verifier zero-knowledge (MVZK) protocol for stateless and deterministic nonce generation. Our protocol can be used to build secure blockchain wallets and custody solutions, enhancing key protection.

Keywords: Multi-Party EdDSA Signing · Multi-Verifier Zero-Knowledge Proof · Threshold Signature · Key Protection.

1 Introduction

Threshold signatures enable a user to distribute its secret key among multiple parties, allowing a quorum of parties, exceeding a predefined threshold, to collaboratively sign a message. This concept was explored in earlier works (see, e.g.,

[Des88,GJKR96,SG98,Sho00,MR01,MOR01]) and has recently garnered significant attention due to its applications in key protection—such as safeguarding users' wallets in blockchain-based systems. Recent research has focused on designing concrete and efficient threshold signature protocols for ECDSA (e.g., [Lin17,LN18,GG18,DKLs18,DKLs19,CCL+19,CGG+20,XAX+21,DKLS24]) and Schnorr signatures (e.g., [KG20,RRJ+22,BHK+24,CKM23,CGRS23,BLSW24]).

The Edwards-curve Digital Signature Algorithm (abbr. EdDSA)[BDL+11] has been widely adopted in practice, particularly in blockchain applications. Several blockchain projects, such as Hyperledger [hyp24], Cardano [Car24], Stellar [Ste24] and Decred [dec24] have integrated EdDSA. Additionally, the cryptocurrency Zcash [Zca24] announced its transition to the Ed25519 variant of EdDSA for the JoinSplit signature in its rpcwallet. Technically, EdDSA is a deterministic variant of the Schnorr signature scheme, designed for twisted Edwards curves. It has been standardized by both NIST [Nat19] and the IRTF [JL17]. The structure of the EdDSA signature is that for each message msg, it derives nonce as $\boldsymbol{r} = \mathsf{PRF}_{\mathsf{dk}}(\mathsf{msg}) \in \{0,1\}^{\ell}$ using a pseudorandom function PRF. Here, dk represents the right half of the hash output $\mathsf{H}(\mathsf{sk})$ associated with the secret key sk. Informally, a signature on a message msg consists of a pair (R, σ), where $R = r \cdot G$ and $\sigma = r + s \cdot \mathsf{H}(R, \mathsf{pk}, \mathsf{msg})$. Here, $r = \sum_{i=1}^{\ell} \boldsymbol{r}[i] \cdot 2^{i-1}$, s is the left half of $\mathsf{H}(\mathsf{sk})$, and $\mathsf{pk} = s \cdot G$ is the public key. EdDSA benefits from deterministic nonce generation, as relying on a source of randomness for signing can be challenging in certain contexts, such as public cloud environments [GKMN21]. Another advantage of EdDSA is its stateless design, meaning it does not require the reliable maintenance of counters, which—when used with PRF-can generate fresh randomness. As noted in [PLD+11,BHH+15,GKMN21], reliably updating the state in practice is challenging, and reusing a counter would result in repeated randomness, compromising security.

In this work, we seek to extend the benefits of deterministic and stateless EdDSA signing to threshold signatures. Recently, NIST announced plans to standardize multi-party threshold EdDSA signatures with stateless and deterministic nonce derivation, motivating the development of concretely efficient threshold EdDSA protocols. Our focus is on the full-threshold setting[7], where all secret key shares are required to sign messages. This includes the two-party case and ensures security even in a malicious setting tolerating any corruption. Thresholdizing EdDSA in a malicious setting is a challenging task [MPSW19,GKMN21]. The difficulty arises from the fact that honest parties must use identical randomness across two protocol executions for the same message, while a malicious adversary may introduce inconsistent randomness across executions. Thus, the key challenge is to ensure the correctness of nonce derivation, i.e., $\boldsymbol{r} = \mathsf{PRF}_{\mathsf{dk}}(\mathsf{msg})$, even in the presence of malicious actors, given the non-linear nature of the function PRF. Without such a guarantee, a malicious adversary could mount a forking attack to recover the secret key [MPSW19]. Therefore, the core challenge now

[7] A full-threshold signature protocol is also referred to as a multi-party signature protocol.

is *deterministically generating an EdDSA signature in the presence of malicious adversaries.*

Two approaches can ensure the correctness of nonce derivation: secure multi-party computation (MPC) and zero-knowledge (ZK) proofs. The MPC approach allows all parties to jointly compute $\boldsymbol{r} = \mathsf{PRF}_{\mathsf{dk}}(\mathsf{msg})$ securely, but it incurs significant costs in terms of communication, computation, and the number of rounds. This is exemplified by the work of Bonte et al. [BST21], which employs the MPC approach for nonce derivation under the assumptions of an honest-majority setting, a malicious adversary, and security with abort. In contrast, the ZK approach has each party P_i locally compute $\boldsymbol{r}_i = \mathsf{PRF}_{\mathsf{dk}_i}(\mathsf{msg})$ using its share dk_i and prove its correctness with a ZK proof. The shares $\{\boldsymbol{r}_i\}_{i\in[n]}$ are then combined into a group element $R = r \cdot G$, where $\mathsf{dk} = \bigoplus_{i\in[n]} \mathsf{dk}_i$, $\boldsymbol{r} = \bigoplus_{i\in[n]} \boldsymbol{r}_i$ with n parties and jointly nonce $r = \sum_{j=1}^{\ell} \boldsymbol{r}[j] \cdot 2^{j-1}$ of ℓ bit length. Compared to the MPC approach, the ZK approach is more efficient.

The NIST call [BP23] supports multi-party threshold EdDSA protocols under the 'pseudorandom per quorum' mode, where nonce derivation is verified using ZK proofs (referred to as Type II in [BP23]). The state-of-art work by Garillot et al.[GKMN21] adopts the zero-knowledge from garbled circuits (ZKGC) paradigm[JKO13] to prove the correctness of nonce derivation, achieving significantly lower communication and fewer rounds compared to the MPC-based protocol [BST21]. Although the multi-party EdDSA protocol in [GKMN21] offers fast computation and requires only three rounds, it still requires high communication costs.

Recent work [KOR23] (resp., [CGG+20]) proposed efficient approaches for designing stateless and deterministic Schnorr signature protocols in the two-party (resp., multi-party) setting. However, these methods rely on customized functions for nonce derivation, making them incompatible with the EdDSA standard, which uses either SHA512 or SHAKE256 to instantiate PRF. Concurrent work by Komlo and Goldberg [KG24] introduced an approach based on verifiable pseudorandom secret sharing to achieve stateless and deterministic nonce derivation. While their method features minimal communication and requires only two rounds, it is limited to the honest-majority setting (i.e., fewer than half the parties can be corrupted).

1.1 Our Contributions

This paper proposes a new stateless and deterministic multi-party EdDSA protocol that remains secure even in the presence of malicious adversaries capable of corrupting any number of parties. Our primary technical contribution is a low-communication approach for designing a multi-verifier zero-knowledge (MVZK) protocol to ensure the stateless and deterministic derivation of EdDSA nonces.

Specifically, we leverage the concept of pseudorandom correlation functions (PCF) with programmability properties [BCG+20,BCG+22,CD23,BCE+23] to generate information-theoretic message authentication codes (IT-MACs) in a stateless and deterministic manner. Building on IT-MACs over $\mathbb{F}_2$, we extend the

Table 1: **Comparison of stateless and deterministic nonce derivation protocols with malicious security in the two-party setting**. $|\mathcal{C}|$ denotes the size of a PRF circuit, κ is the computational security parameter, s is the statistical security parameter, $|q|$ represents the bit-length of an element in $\mathbb{Z}_q$, ℓ is the bit length of PRF outputs and m is a large parameter used in [KOR23]. Concrete costs are given for $|\mathcal{C}| = 58K$, $\kappa = 128$, $s = 60$, $|q| = 256$, $\ell = 512$ and $m = 3597$ when thresholdizing HashEdDSA [Nat19] over the Edwards curve Ed25519, where PRF is instantiated by $\mathsf{SHA512}$. SRSA denotes the strong RSA assumption, DL represents the discrete-logarithm assumption, CRHF denotes collision-resistant hash function and LPN denotes the learning parity with noise assumption.

Protocols	EdDSA PRF	Asymptotic Comm.	Concrete Comm.	Rounds	Assumptions
[NRSW20]	no	$O(\|q\|)$	1.1 KB	2	DDH
[KOR23]	no	$O(m + \|q\| + \kappa)$	0.88 KB	1	DCR+SRSA
[GKMN21]	yes	$O(\|\mathcal{C}\|\kappa + \|q\|\kappa)$	1.01 MB	3	PRF+CRHF
This work	yes	$O(\|\mathcal{C}\| + \log(\|\mathcal{C}\|)\kappa + \ell\kappa)$	32.47 KB	2	LPN

VOLE-based ZK protocol [BMRS21] from the single-verifier to the multi-verifier setting, enabling each party P_i to prove $\boldsymbol{r}_i = \mathsf{PRF}_{\mathsf{dk}_i}(\mathsf{msg})$. The MVZK protocol for nonce derivation can be made non-interactive, stateless, and deterministic through the Fiat-Shamir (FS) transformation. We then convert IT-MACs over $\mathbb{F}_2$ into IT-MACs over $\mathbb{Z}_q$, where q is a prime, by generalizing and refining the edaBits technique [EGK+20]. Specifically, we extend edaBits from the MPC setting to the MVZK setting and enhance the underlying check protocol using a 'sacrificing' technique, ensuring that only one correct edaBit is needed per signing. Subsequently, we locally convert IT-MACs over $\mathbb{Z}_q$ into IT-MACs over an elliptic-curve group $\mathbb{G}$ without requiring communication, following the observations in [STA19,KOR23]. As a result, each party obtains an IT-MAC for the group element $R_i = r_i \cdot G$. These IT-MACs are then opened to compute the correct group element $R = \sum_{i\in[n]} R_i = r \cdot G$, where $r_i \in \mathbb{Z}_q$ is the arithmetic representation of $\boldsymbol{r}_i$, $r = \sum_{i\in[n]} r_i$, and n is the number of parties. For further technical details, please refer to Section 3.

Comparison of two-party protocols for stateless and deterministic nonce derivation. For the two-party case, Table 1 compares our protocol with existing protocols for deterministic, stateless, and verifiable nonce derivation in the malicious setting. The concrete communication cost is calculated for statelessly and deterministically generating $R = r \cdot G$, where $r \in \mathbb{Z}_q$ is the arithmetic representation of $\mathsf{PRF}_{\mathsf{dk}}(\mathsf{msg})$.

Both works by Nick et al.[NRSW20] and Kondi et al.[KOR23] achieve significantly lower communication costs than other protocols. The protocol by Kondi et al.[KOR23] achieves the optimal one-round execution. However, these approaches rely on customized functions for nonce derivation rather than the

Table 2: **Comparison between our protocol and the state-of-the-art protocol for generating multi-party EdDSA signatures statelessly and deterministically.** n denotes the total number of parties. Concrete communication costs are given for $|\mathcal{C}| = 58K$, $\kappa = 128$, $|q| = 256$ and $\ell = 512$, when thresholdizing HashEdDSA [Nat19] over the Edwards curve Ed25519 among three parties and five parties.

Protocols	Asymptotic Communication	Comm. Cost ($n = 3$)	Comm. Cost ($n = 5$)	Rounds
[GKMN21]	$O(n^2(\|\mathcal{C}\|\kappa + \|q\|\kappa))$	10.76 MB	35.89 MB	3
This work	$O(n^2(\|\mathcal{C}\| + \log\|\mathcal{C}\|\kappa + \ell\kappa))$	0.19 MB	0.63 MB	3

PRF function used in the EdDSA standard. The prior work by Garillot et al.[GKMN21], which is closest to ours, employs the ZKGC approach to prove $\boldsymbol{r} = \mathsf{PRF}_{\mathsf{dk}}(\mathsf{msg})$. Our nonce-derivation protocol leverages the recent LPN-based PCF construction [BCG+22] and achieves approximately a 31.9× reduction in communication cost compared to the protocol by Garillot et al.[GKMN21] in the two-party setting. As such, our work represents a trade-off between efficiency and the functionality supported by NIST[BP23].

Comparison of multi-party EdDSA signature protocols with malicious security. Table 2 compares our multi-party EdDSA signature protocol with the state-of-the-art protocol [GKMN21], both of which are stateless and deterministic. We focus solely on the signing phase, as the key generation phase is executed only once. The primary cost of multi-party EdDSA signature protocols lies in generating $R = r \cdot G$. A detailed comparison of communication costs and rounds for the two-party case between our protocol and [GKMN21] is provided in Table 1. Thus, Table 2 focuses on cases involving more than two parties, specifically considering $n = 3$ and $n = 5$. Both our protocol and that of [GKMN21] are secure in the dishonest-majority setting (i.e., with a corruption threshold of $t = n - 1$). We do not include a comparison with the protocols in [BST21,KG24], which are designed for the honest-majority setting (i.e., $t = \lfloor (n-1)/2 \rfloor$).

Compared to the state-of-the-art protocol [GKMN21], our protocol reduces the communication cost by a factor of 56× for both $n = 3$ and $n = 5$, while maintaining the same number of rounds. As a trade-off, the computational cost increases by approximately 2.25×. The communication cost is evaluated for HashEdDSA over the Edwards curve Ed25519. In HashEdDSA, msg represents the hash digest of the original message, while in another EdDSA variant, $\mathsf{PRF}_{\mathsf{dk}}(\mathsf{msg})$ is computed directly on the original message msg. For this variant, the circuit size required to compute $\mathsf{PRF}_{\mathsf{dk}}(\mathsf{msg})$ increases significantly, especially for longer messages. In such cases, our protocol would achieve even greater communication improvements. While both our protocol and [GKMN21] have $O(n^2)$ communication complexity, we leverage a simple binary tree optimization from [QYYZ22] to reduce the communication complexity to $O(n)$, at the cost of requiring an additional $O(\log n)$ rounds (see Section 4.2 for more de-

tails).[8] Therefore, one practical application of our protocol is the construction of secure cryptocurrency wallets, where key shares are distributed across multiple devices, making them difficult to steal. Another application is bandwidth-limited scenarios (e.g., mobile computing, Internet of Things, etc.), making protocols with significantly lower communication costs (e.g., our protocol) advantageous for achieving faster running times.

2 Preliminaries

2.1 Notation

Let κ be the security parameter and n be the number of parties. We denote by $[n]$ the set $\{1, \ldots, n\}$ and $[a, b]$ the set $\{a, \ldots, b\}$. Bold lower-case letters, e.g., $\boldsymbol{x}$, denote the vectors, and $\boldsymbol{x}[i]$ is the i-th element of $\boldsymbol{x}$ with $\boldsymbol{x}[1]$ as the first entry and $\boldsymbol{x}[a:b]$ as the sub-vector $\{\boldsymbol{x}[a], \ldots, \boldsymbol{x}[b]\}$. Let $\mathbb{G}$ be an additive cycle group of generator G and order q, and upper-case letters, e.g., X, denote the group element. For a circuit $\mathcal{C}$, we use $|\mathcal{C}|$ to denote the number of multiplication gates. We use $[\![x]\!]_2$ denote the *multi-verifier* authenticated share of x over $\mathbb{F}_{2^\kappa}$ and $[\![x]\!]_q$ denotes the *multi-verifier* authenticated share of x over $\mathbb{Z}_q$. We use $P_1, \ldots, P_n$ to denote n parties, $\mathcal{P}$ to denote the prover and $\mathcal{V}_1, \ldots, \mathcal{V}_N$ to denote N verifiers. For multi-party signature, we let $N = n - 1$.

2.2 Security Model

Universal Composability We prove the security of our protocols in the *universal composability* (UC) framework [Can01] against a static, malicious adversary who corrupts up to $n-1$ out of n parties. We say that a protocol Π *UC-realizes* an ideal functionality $\mathcal{F}$ if for any probabilistic polynomial time (PPT) adversary $\mathcal{A}$, there exists a PPT adversary (called the simulator) $\mathcal{S}$ such that for any PPT environment $\mathcal{Z}$ with arbitrary auxiliary input z, the output distribution of $\mathcal{Z}$ in the *real-world* execution where the parties interact with $\mathcal{A}$ and execute Π is computationally indistinguishable from the output distribution of $\mathcal{Z}$ in the *ideal-world* execution where the parties interact with $\mathcal{S}$ and $\mathcal{F}$. Environment $\mathcal{Z}$ is a powerful entity with total control over adversary $\mathcal{A}$ and can choose the inputs and see the outputs of all parties.

Security in the presence of malicious adversaries We use $P_1, \ldots, P_n$ to denote n parties and $\mathcal{I}$ to denote the set of corrupted parties. In this paper, we consider *security with abort*, meaning that a corrupted party can obtain output while the honest party does not. In this case, the ideal-world adversary receives output first and then sends either $(\mathsf{continue}, i)$ or (abort, i) to the ideal functionality, for $i \notin \mathcal{I}$ to instruct the functionality either to deliver the output to P_i or to send abort to P_i. We always assume that output is sent this way and omit it hereafter for the sake of simplicity.

[8] It remains unclear how to apply the tree-based communication optimization to the previous protocol[GKMN21], which is based on ZKGC.

2.3 EdDSA Signature Algorithm

This section provides details on EdDSA. According to the NIST and IRTF standards [JL17,Nat19], EdDSA has two variants based on how randomness is generated: in the first variant, $\boldsymbol{r} = \mathsf{PRF}(\mathsf{dk}, \mathsf{msg})$, and in the second, $\boldsymbol{r} = \mathsf{PRF}(\mathsf{dk}, \mathsf{H}(\mathsf{msg}))$. This paper focuses on the second variant, as the PRF circuit processes a fixed message length in this case. For simplicity, we denote $\mathsf{H}(\mathsf{msg})$ as msg, which has minimal impact when the message is publicly known. EdDSA offers two versions, based on the Edwards curves Ed25519 and Ed448. This paper focuses on Ed25519, implemented with SHA-512 and $\ell_b = 256$, a widely adopted configuration for EdDSA-based applications. Our multi-party EdDSA signature protocol is also compatible with Ed448, which uses SHAKE256 and $\ell_b = 456$. Formally, EdDSA includes three core algorithms (i.e., **KeyGen, Sign, Verify**) and the security of plain EdDSA has been proven recently in [BCJZ21,BDD23].

Parameters: EdDSA is parameterized by $\mathtt{params} = (\mathbb{E}_p, \mathbb{G}, q, G, \ell_b, \ell, \mathsf{H}_{\mathsf{sig}}, \mathsf{PRF})$, where $\mathbb{E}_p$ is the twisted elliptic curve, $\mathbb{G}$ is an additive cycle group of generator G and order q, ℓ_b is the bit-length of secret EdDSA scalars, $\mathsf{PRF} : \{0,1\}^{\ell_b+\ell} \to \{0,1\}^{\ell}$ is a pseudorandom function with ℓ-bit output (satisfying $\ell = 2\ell_b$) and $\mathsf{H}_{\mathsf{sig}} : \{0,1\}^* \to \mathbb{Z}_q$ is a hash-to-integer function.

KeyGen($\mathtt{params}$):

1. Sample a secret key $\mathsf{sk} \leftarrow \{0,1\}^{\ell_b}$ of ℓ_b bit length, and compute a hash value $(\boldsymbol{h}[1], \boldsymbol{h}[2], \ldots, \boldsymbol{h}[2\ell_b]) := \mathsf{PRF}(\mathsf{sk})$.
2. Assign $\boldsymbol{h}[1] = \boldsymbol{h}[2] = \boldsymbol{h}[3] = \boldsymbol{h}[\ell_b] = 0$, $\boldsymbol{h}[\ell_b - 1] = 1$. Then use the updated vector $\boldsymbol{h}[1 : \ell_b]$ to define a secret scalar $s \in \mathbb{Z}_q$ i.e., $s = \sum_{i=1}^{\ell_b} \boldsymbol{h}[i] \cdot 2^{i-1} \mod q$, and use the higher second half $\boldsymbol{h}[\ell_b + 1 : 2\ell_b]$ as the derived key dk.
3. Compute the public key $\mathsf{pk} = s \cdot G$.

Sign($\mathsf{dk}, s, \mathsf{pk}, \mathsf{msg}$):

1. Derive pseudorandom value as $\boldsymbol{r} = \mathsf{PRF}(\mathsf{dk}, \mathsf{msg})$ and compute $r = \sum_{i=1}^{\ell} 2^{i-1} \cdot \boldsymbol{r}[i] \mod q$.
2. Compute $R = r \cdot G$, $h = \mathsf{H}_{\mathsf{sig}}(R, \mathsf{pk}, \mathsf{msg})$ and $\sigma = r + h \cdot s \mod q$.
3. Output a signature (R, σ).

Verify($\mathsf{pk}, \mathsf{msg}, (R, \sigma)$):

1. Compute $h' = \mathsf{H}(R, \mathsf{pk}, \mathsf{msg})$.
2. Output 1 (accept) iff $(2^3 \cdot \sigma) \cdot G = 2^3 \cdot R + (2^3 \cdot h') \cdot \mathsf{pk}$ holds; otherwise, output 0 (reject).

2.4 Functionality of Commitment $\mathcal{F}_{\mathsf{Com}}$

To realize multi-party EdDSA signing, we use an ideal commitment functionality $\mathcal{F}_{\mathsf{Com}}$, formally defined in Fig. 1.

Functionality $\mathcal{F}_{\mathsf{Com}}$

This functionality runs with parties $P_1, \ldots, P_n$, as follows:

- Upon receiving $(\mathsf{commit}, sid, i, x)$ from a party P_i (for $i \in [n]$), record $(\mathsf{commit}, sid, i, x)$ and send $(\mathsf{receipt}, sid, i)$ to all the other parties. If some $(\mathsf{commit}, sid, i, *)$ is already recorded, ignore the message.
- Upon receiving $(\mathsf{test}, sid, j, k)$ from a corrupted party P_j, if $(\mathsf{commit}, sid, i, x)$ has been stored then send $(\mathsf{reveal}, sid, j, x_k)$ to P_j, where x_k is k-th bit of committed value x.
- Upon receiving $(\mathsf{decommit}, sid, i)$ from a party P_i (for $i \in [n]$): if $(\mathsf{commit}, sid, i, x)$ has been stored then send $(\mathsf{decommit}, sid, i, x)$ to all the other parties.

Fig. 1: The Commitment Functionality

In our protocol, all the inputs to be committed have a sufficient high entropy. In this case, we can securely realize $\mathcal{F}_{\mathsf{Com}}$ by defining $\mathsf{Com}(x) = \mathsf{H}(x)$ for high-entropy input, where $\mathsf{H}(\cdot) : \{0,1\}^* \to \{0,1\}^{2\kappa}$ is a cryptographic hash function with security parameter κ. Thus, in this case, all other parties can test guesses on some bits of input after it has been committed. This has no impact on security, as the simulation of commitments and the extraction of inputs still work in the random-oracle model.

2.5 Functionality of Committed NIZK $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{DL}}}$

Fig. 2 overviews the committed non-interactive zero-knowledge proof functionality for discrete logarithm relation, denoted as $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{DL}}}$.

Functionality $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{DL}}}$

This functionality runs with parties $P_1, \ldots, P_n$, as follows:

- Upon receiving $(\mathsf{com\text{-}prove}, sid, Q, x)$ from a party P_i (for $i \in [n]$), if $Q \neq x \cdot G$ or sid has been previously used then ignore the message. Otherwise, store (sid, i, Q) and send $(\mathsf{proof\text{-}receipt}, sid)$ to the other parties.
- Upon receiving $(\mathsf{decom\text{-}proof}, sid)$ from a party P_i (for $i \in [n]$): (sid, i, Q) has been stored then send $(\mathsf{decom\text{-}proof}, sid, Q)$ to the other parties.

Fig. 2: The Committed NIZK Functionality for DL Relation

The NIZK proof of knowledge in the random oracle model can be implemented following several multi-party signing protocols, such as related works

[Lin17,LN18,DKLs18,DKLs19]. In this paper, we employ the standard Schnorr proof to demonstrate knowledge of the discrete logarithm of an elliptic curve point. Note that the function $\mathsf{H}(\cdot) : \{0,1\}^* \to \mathbb{Z}_q$ used in Schnorr is a cryptographic hash-to-integer function. This protocol can be converted into a non-interactive version using the Fiat-Shamir heuristic [FS87]. By combining the above instantiation of $\mathcal{F}_{\mathsf{Com}}$ with the Schnorr proof, we obtain an efficient implementation of the $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{DL}}}$ functionality, which will be applied in the key generation phase of the multi-party EdDSA signing protocol. Additionally, since a 'high-entropy random source' is available during the key generation phase of EdDSA, this instantiation does not affect our contribution.

2.6 Pseudorandom Correlation Function

The Pseudorandom Correlation Function (PCF) was first introduced by Boyle et al.[BCG+20] based on the Learning Parity with Noise (LPN) assumption and has since been further studied [BCG+22,CD23]. It enables the incremental and on-demand local generation of an arbitrary polynomial number of pseudorandom correlations from a pair of short correlated keys. Below, we define PCF-based Vector Oblivious Linear Evaluation (VOLE) correlations.

Definition 1. *Let* $1 \leq \tau_0(\kappa), \tau_1(\kappa) \leq \mathsf{poly}(\kappa)$ *be the output-length functions, and let* $\mathcal{M} \subseteq \mathbb{F}$ *be a set of allowed master keys for verifier. Let* $(\mathsf{Setup}, \mathcal{Y})$ *be probabilistic algorithms such that:*

- $\mathsf{Setup}(1^\kappa, \mathcal{M})$ *sample a master secret key from* $\mathcal{M}$*, for example,* $\mathsf{msk} := \Delta$*;*
- $\mathcal{Y}(1^\kappa, \mathsf{msk})$ *return a pair of outputs* $(y_0, y_1) \in \{0,1\}^{\tau_0(\kappa)} \times \{0,1\}^{\tau_1(\kappa)}$*, defining a correlation on the outputs.*

We say that $(\mathsf{Setup}, \mathcal{Y})$ *define a reverse sampleable correlation, if there exists a probabilistic polynomial time (*PPT*) algorithm* $\mathsf{RSample}$ *such that*

- $\mathsf{RSample}(1^\kappa, \mathsf{msk}, b \in \{0,1\}, y_b \in \{0,1\}^{\tau_b(\kappa)})$ *return* $y_{1-b} \in \{0,1\}^{\tau_{1-b}(\kappa)}$*, such that for all* $\mathsf{msk} \in \mathcal{M}$ *and* $b \in \{0,1\}$ *the following distributions are statistically close:*
  - $\{(y_0, y_1) | (y_0, y_1 \leftarrow \mathcal{Y}(1^\kappa, \mathsf{msk})\}$ *and*
  - $\{(y_0, y_1) | (y_0^*, y_1^* \leftarrow \mathcal{Y}(1^\kappa, \mathsf{msk}), y_b \leftarrow y_b^*, y_{1-b} \leftarrow \mathsf{RSample}(1^\kappa, \mathsf{msk}, b, y_b^*)\}$

To show how this reverse sampling definition works, we adopt the distribution for vector oblivious linear evaluation (VOLE) correlations if $\mathcal{Y}(1^\kappa, \Delta)$ samples $x \leftarrow \mathbb{F}, k \leftarrow \mathbb{F}_p$, computes $m = k + x \cdot \Delta \in \mathbb{F}_p$ and outputs $((x, m), k)$, where $\mathbb{F}$ could be $\mathbb{F}_2$ (with $p = 2^\kappa$) or $\mathbb{Z}_q$ (with $p = q$).

Definition 2. *Let* $(\mathsf{Setup}, \mathcal{Y})$ *fix a reverse-sampleable correlation with setup which has output length functions* $\tau_0(\kappa), \tau_1(\kappa)$ *and sets* $\mathcal{M}$ *of allowed master keys, and let* $\kappa \leq n(\kappa) \leq \mathsf{poly}(\kappa)$ *be an input length function. Let* $(\mathsf{PCF.Gen}, \mathsf{PCF.Eval})$ *be a pair of algorithms with the following syntax:*

- $\mathsf{PCF.Gen}(1^\kappa, \mathsf{msk})$ *is a* PPT *algorithm that outputs a pair of keys* (k_0, k_1)*;*

$\underline{\textbf{Experiment } \exp_0^{pr}(\kappa)}$

for $i = 1$ **to** $Q(\kappa)$:

$v^i \leftarrow \{0,1\}^{n(\kappa)}$
$\underline{(y_0^{(i)}, y_1^{(i)}) \leftarrow \mathcal{Y}(1^\kappa, \mathsf{msk})}$

$b \leftarrow \mathcal{A}(1^\kappa, (v^i, y_0^{(i)}, y_1^{(i)})_{i \in [Q(\kappa)]})$
return b

Fig. 3: Correlated outputs of the $\mathcal{Y}$-function

$\underline{\textbf{Experiment } \exp_1^{pr}(\kappa)}$

$(k_0, k_1) \leftarrow \mathsf{PCF.Gen}(1^\kappa, \mathsf{msk})$
for $i = 1$ **to** $Q(\kappa)$:

$v^i \leftarrow \{0,1\}^{n(\kappa)}$
$\underline{\textbf{for } b \leftarrow \{0,1\}\colon y_b^{(i)} \leftarrow \mathsf{PCF.Eval}(b, k_b, v^{(i)})}$

$b \leftarrow \mathcal{A}(1^\kappa, (v^i, y_0^{(i)}, y_1^{(i)})_{i \in [Q(\kappa)]})$
return b

Fig. 4: Pseudorandom $\mathcal{Y}$-correlated outputs of a PCF

$\underline{\textbf{Experiment } \exp_0^{sec}(\kappa)}$

$(k_0, k_1) \leftarrow \mathsf{PCF.Gen}(1^\kappa, \mathsf{msk})$
for $i = 1$ **to** $Q(\kappa)$:

$v^i \leftarrow \{0,1\}^{n(\kappa)}$
$\underline{y_{1-b}^{(i)} \leftarrow \mathsf{PCF.Eval}(1-b, k_{1-b}, v^{(i)})}$

$b \leftarrow \mathcal{A}(1^\kappa, k_b, (v^i, y_{1-b}^{(i)})_{i \in [Q(\kappa)]})$
return b

Fig. 5: Output distributions of a PCF

– $\mathsf{PCF.Eval}(b, k_b, v)$ *is a deterministic polynomial-time algorithm that on input* $b \in \{0,1\}$*, key* k_b *and value* $v \in \{0,1\}^{n(\kappa)}$*, outputs a value* $y_b \in \{0,1\}^{\tau_b(\kappa)}$

The $(\mathsf{PCF.Gen}, \mathsf{PCF.Eval})$ (in Definition 2) is a (weak) pseudorandom correlation function (PCF) for $\mathcal{Y}$, if the following conditions hold:

Experiment $\exp_1^{sec}(\kappa)$

$k_b \leftarrow \mathcal{S}_b(1^\kappa, \mathsf{msk})$
for $i = 1$ **to** $Q(\kappa)$:

$v^i \leftarrow \{0,1\}^{n(\kappa)}$
$y_b^{(i)} \leftarrow \mathsf{PCF.Eval}(b, k_b, v^{(i)})$
$y_{1-b}^{(i)} \leftarrow \mathsf{RSample}(1^\kappa, \mathsf{msk}, b, y_b^{(i)})$

$b \leftarrow \mathcal{A}(1^\kappa, k_b, (v^i, y_{1-b}^{(i)})_{i\in[Q(\kappa)]})$
return b

Fig. 6: Output distributions with RSample algorithm as in Definition 1

– **Pseudorandom $\mathcal{Y}$-correlated outputs**. For every $\mathsf{msk} \in \mathcal{M}$, and non-uniform adversary $\mathcal{A}$ of size $\mathsf{poly}(\kappa)$, and every $Q = \mathsf{poly}(\kappa)$, it holds that

$$|\Pr[\exp_0^{pr}(\kappa) = 1| - |\Pr[\exp_1^{pr}(\kappa) = 1| \leq \mathsf{negl}(\kappa)$$

for all sufficiently large κ, where $\exp_b^{pr}(\kappa)$ for $b \in \{0,1\}$ is defined in Fig. 3 and Fig. 4 (with $Q(\kappa)$ samples given access to $\mathcal{A}$).
– **Security**. For each $b \in \{0,1\}$ there is a simulator $\mathcal{S}_b$ such that for every $\mathsf{msk} \in \mathcal{M}$, any every non-uniform adversary $\mathcal{A}$ of size $B(\kappa)$, and every $Q = \mathsf{poly}(\kappa)$, it holds that

$$|\Pr[\exp_0^{sec}(\kappa) = 1| - |\Pr[\exp_1^{sec}(\kappa) = 1| \leq \mathsf{negl}(\kappa)$$

for all sufficiently large κ, where $\exp_b^{sec}(\kappa)$ for $b \in \{0,1\}$ is defined in Fig. 5 and Fig. 6 (again, with $Q(\kappa)$ samples).

To simplify, we define "PCF assumption" which means the underlying PCF primitive satisfies the standard security property defined by Boyle et al. [BCG+20] and PCF can be instantiated under the LPN assumption [BCG+22,CD23]. Thus, our protocol essentially relies on the LPN assumption (as discussed in Table 1).

2.7 Multi-Verifier Programmable PCF

Fig. 7 illustrates a multi-verifier extension of the two-party PCF described in Section 2.6.

In the multi-verifier setting, $\mathcal{P}$ generates a VOLE correlation with each verifier $\mathcal{V}_i$, for $i \in [N]$, such that $\mathcal{P}$ obtains the same $\boldsymbol{x} \in \mathbb{F}^\ell$, and each $\mathcal{V}_i$ receives the same $\Delta^i \in \mathbb{F}$ across all VOLE correlations. Existing PCF schemes satisfy the programmability property defined in [BCG+19] and have been used in [BCG+22] so that one value can be programmed to be the same across multiple PCF instances. Specifically, a multi-verifier programmable PCF takes additional

Macro Multi-verifier PCF

$\mathsf{PCF.Gen}_{\mathsf{mv}}$ $(1^\kappa, \mathbb{F})$ runs N executions of $\mathsf{PCF.Gen}(1^\kappa, \mathsf{msk})$ to generate $(k_0, k_1, \ldots, k_N)$ such that the size of every seed k_i is at most $O_\kappa(N \log^2(\ell))$. In particular, $\mathsf{PCF.Gen}_{\mathsf{mv}}$ (1^κ) executes as follows:

1. For each $i \in [N]$, sample $\Delta^i \leftarrow \mathbb{F}_p$.
2. For each $i \in [N]$, run $\mathsf{PCF.Gen}(1^\kappa, \Delta^i)$ to generate a pair of seeds (k_0^i, k_1^i).
3. For each $i \in [N]$, output $k_i := k_1^i$ to $\mathcal{V}_i$ and $k_0 := \{k_0^i\}_{i \in [N]}$ to $\mathcal{P}$.

$\mathsf{PCF.Eval}_{\mathsf{mv}}$ (i, k_i, v) runs N executions of $\mathsf{PCF.Eval}$ to generate parties' shares on a vector of multi-verifier authenticated sharing $[\![\boldsymbol{x}]\!]_2$. For each $i \in [N]$, $\mathsf{PCF.Eval}_{\mathsf{mv}}$ (i, k_i) performs the following:

1. For each $i \in [N]$, run $\mathsf{PCF.Eval}(1, k_1^i, v)$ to generate $y_1^i = \{\boldsymbol{k}^{(i)}, \Delta^i\}$.
2. For each $i \in [N]$, run $\mathsf{PCF.Eval}(0, k_0^i, v)$ to generate $y_0^i = \{\boldsymbol{x}, \boldsymbol{m}^{(i)}\}$ such that $\boldsymbol{m}^{(i)} = \boldsymbol{k}^{(i)} + \boldsymbol{x} \cdot \Delta^i$.

Fig. 7: Multi-verifier PCF scheme

input $(\boldsymbol{x}, \Delta^i)$ from one single $\mathcal{P}$ and multiple verifier $\mathcal{V}_i$, and outputs seeds that expand into the multi-verifier VOLE correlations with *fixed* $\boldsymbol{x}$ and Δ^i. Using this programmability, one can extend PCF from the two-party setting to the multi-party setting (see [BCG+22] for details). Building on the multi-verifier programmable PCF, we present the construction of multi-verifier authenticated sharing, as defined in Section 3.1, while ensuring security.

3 Technical Overview

This section provides a technical overview of our work, with detailed descriptions and security proofs deferred to later sections. Fig. 8 outlines the key techniques in our designed protocols. Specifically, we define IT-MACs over a group and leverage them to design a non-interactive, stateless, and deterministic multi-verifier zero-knowledge proof (MVZK) for nonce derivation. Finally, we present a multi-party EdDSA signing protocol that achieves optimal communication efficiency.

3.1 Multi-Verifier IT-MACs over Groups

We begin by defining the concept of multi-verifier *information-theoretic message authentication codes (IT-MACs)*, which generalizes the single-verifier VOLE-based ZK protocol [BMRS21]. The values to be authenticated are in either $\mathbb{F}_2$ or $\mathbb{F}_{2^\kappa}$, with authentication performed over the binary extension field $\mathbb{F}_{2^\kappa}$. Specifically, let $\mathcal{V}_1, \ldots, \mathcal{V}_N$ represent N verifiers, and let $\Delta^i \in \mathbb{F}_{2^\kappa}$ be a uniform *global key* known only to each verifier $\mathcal{V}_i$. A value $x \in \mathbb{F}_2$ or $\mathbb{F}_{2^\kappa}$, known by the

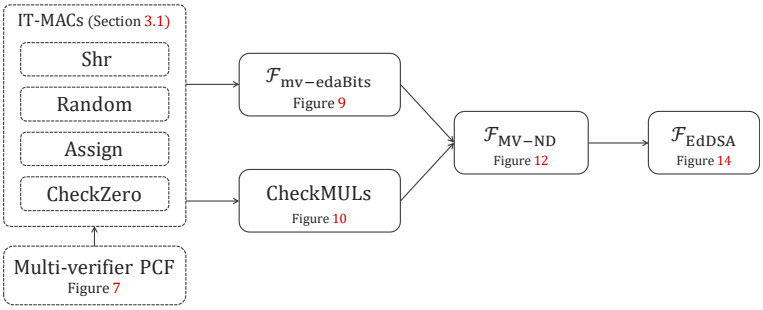

Fig. 8: Technique Outline

prover $\mathcal{P}$, is authenticated as $[\![x]\!]_2 = (x, m_1, \ldots, m_N), k_1, \ldots, k_N$, where each $m_i = k_i + x \cdot \Delta^i \in \mathbb{F}_{2^\kappa}$, with the same value of x. Each verifier $\mathcal{V}_i$ holds a *local MAC key* $k_i, i \in [N]$, while the prover $\mathcal{P}$ retains the secret value along with the corresponding *MAC tags* $(x, m_1, \ldots, m_N)$. Additionally, we extend this notation to vectors, arithmetic operations, and groups of authenticated values as follows:

- **Multi-Verifier IT-MACs over Vectors**. Let $[\![\boldsymbol{x}]\!]_2 = \{(\boldsymbol{x}, \boldsymbol{m}_1, \ldots, \boldsymbol{m}_N), \boldsymbol{k}_1, \ldots, \boldsymbol{k}_N\}$ as the multi-verifier IT-MACs over vectors such that $\mathcal{P}$ holds $\boldsymbol{x} \in \mathbb{F}_2^\ell, \boldsymbol{m}_1, \ldots, \boldsymbol{m}_N \in \mathbb{F}_{2^\kappa}^\ell$ while $\mathcal{V}_i$ holds global key $\Delta^i \in \mathbb{F}_{2^\kappa}$ and local MAC key $\boldsymbol{k}_i \in \mathbb{F}_{2^\kappa}^\ell$ with $\boldsymbol{m}_i = \boldsymbol{k}_i + \Delta^i \cdot \boldsymbol{x} \in \mathbb{F}_{2^\kappa}$.
- **Multi-Verifier IT-MACs over Arithmetic**. Let $[\![x]\!]_q = \{(x, m_1, \ldots, m_N), k_1, \ldots, k_N\}$ as the multi-verifier IT-MACs over arithmetic operations such that $\mathcal{P}$ holds $x, m_1, \ldots, m_N \in \mathbb{Z}_q$ and $\mathcal{V}_i$ holds global key $\Lambda^i \in \mathbb{Z}_q$ and local MAC key $k_i \in \mathbb{Z}_q$ with $m_i = k_i + \Lambda^i \cdot x \mod q$.
- **Multi-Verifier IT-MACs over Groups**. Let $[\![X]\!]_q = \{(X, M_1, \ldots, M_N), K_1, \ldots, K_N\}$ as the multi-verifier IT-MACs over groups such that $\mathcal{P}$ holds $X, M_1, \ldots, M_N \in \mathbb{G}$ while $\mathcal{V}_i$ holds glocal key $\Lambda^i \in \mathbb{Z}_q$ and local MAC key $K_i \in \mathbb{G}$ with $M_i = K_i + \Lambda^i \cdot X \in \mathbb{G}$.

All authenticated values support additive homomorphism. For instance, given authenticated bits over $\mathbb{F}_2$, i.e., $[\![x_1]\!]_2, \ldots, [\![x_\ell]\!]_2$ and public coefficients $c_1, \ldots, c_\ell$, $c \in \mathbb{F}_{2^\kappa}$, the parties can locally compute $[\![y]\!]_2 = \sum_{i=1}^{\ell} c_i \cdot [\![x_i]\!]_2 + c$. Below, we define four macros that will be used throughout this paper.

Random. To generate an authenticated value $[\![r]\!]_2$, where $r \in \mathbb{F}_2$ is a uniformly random value, $\mathcal{P}$ and the verifiers $\mathcal{V}$s can invoke $\mathsf{PCF.Eval}_{\mathsf{mv}}$ (c.f. Fig. 7 in Section 2.7) in a stateless and deterministic manner, using precomputed k_0 for $\mathcal{P}$ and k_i for each $\mathcal{V}_i$, where $i \in [N]$. We use Random to denote this macro.

Assign. Given an input $x \in \mathbb{F}_2$ from $\mathcal{P}$, $\mathcal{P}$ and the verifiers $\mathcal{V}$s execute $[\![r]\!]_2 \leftarrow \mathsf{Random}$. Then, $\mathcal{P}$ sends $y = r - x \in \mathbb{F}_2$ to the verifiers, and all parties compute $[\![x]\!]_2 = [\![r]\!]_2 + y$. We denote this assignment procedure as $\mathsf{Assign}(x)$.

Shr. Given an input $\boldsymbol{x} \in \mathbb{F}_2^\ell$ from $\mathcal{P}$, the protocol simply invokes $\mathsf{Assign}(\boldsymbol{x})$ ℓ times in parallel to generate $[\![\boldsymbol{x}]\!]_2$. We denote this sharing procedure as $\mathsf{Shr}(x)$. Note that the Shr macro is used only during key generation with a reliable source of randomness.

Checking Zero. An authenticated value $[\![x]\!]_2$ can be checked for $x = 0$ by having $\mathcal{P}$ send m_i to the corresponding verifier $\mathcal{V}_i$, who then verifies whether $m_i = k_i$. We denote this verification process with the macro $\mathsf{CheckZero}([\![x]\!]_2)$.

In this case, a broadcast is not required, which means a malicious $\mathcal{P}$ could send the correct (m_i) to verifier $\mathcal{V}_i$, while sending incorrect (m_j) to verifier $\mathcal{V}_j$, resulting in $m_i = k_i$, but $m_j \neq k_j$. This could cause $\mathcal{V}_i$ to continue while $\mathcal{V}_j$ aborts. All verifiers can resolve this by announcing their responses and checking for consistency. As long as at least one verifier is honest, any inconsistency will be detected. Since any EdDSA signature generated by our secure signing protocol must be verified by the original EdDSA verification algorithm (see Section 2.3), we eliminate the need for broadcast channels, reducing the communication rounds to $O(1)$. Consequently, the security follows that of two-party IT-MACs and PCF scheme.

3.2 Multi-Verifier Stateless Deterministic Nonce Derivation

The previous section explains why a stateless and deterministic approach is essential. Recent works [NRSW20,GKMN21,KOR23] have contributed to zero-knowledge proof frameworks, all following a similar paradigm outlined below.

1. In the key generation phase, the prover $\mathcal{P}$ commits to the same nonce derivation key dk using verifiable commitments. Specifically, $\mathcal{P}$ commits to each bit of dk, while the verifier $\mathcal{V}$ (i.e., the party responsible for checking the correctness of the nonce derivation) holds the authentication keys.
2. Subsequently, in the signing phase, $\mathcal{P}$ proves an unbounded number of statements (i.e., the correct evaluation of the PRF and exponentiation circuit). Specifically, $\mathcal{P}$ and $\mathcal{V}$ jointly evaluate the target circuit while masking the inputs with the committed nonce derivation keys. If $\mathcal{P}$ uses the correct dk, they must open the same nonce $R = \mathsf{PRF}_{\mathsf{dk}}(\mathsf{msg}) \cdot G$ for each message msg.

Unlike previous works, we aim to simultaneously prove nonce derivation to multiple verifiers. The parties first evaluate the PRF circuit gate by gate. Since all wire values are authenticated using the defined multi-verifier IT-MACs, the ADD gates can be computed locally, while the MULT gates are jointly processed by invoking the $\mathsf{Assign}(\omega_\alpha \cdot \omega_\beta)$ macro. Next, all parties prove the correctness of t multiplication triples $\{\omega_{\alpha,j}, \omega_{\beta,j}, \omega_{\gamma,j}\}_{j\in[t]}$. Here, we adopt the polynomial-based batch verification technique from [BMRS21] and extend it to a multi-verifier setting. First, let's organize these multiplication triples into a $2 \times \frac{t}{2}$ matrix, i.e.,

$$\begin{matrix} \omega_{\alpha,1} & \omega_{\alpha,2} & \omega_{\alpha,3} & \dots & \omega_{\alpha,\frac{t}{2}} \\ \omega_{\alpha,\frac{t}{2}+1} & \omega_{\alpha,\frac{t}{2}+2} & \omega_{\alpha,\frac{t}{2}+3} & \dots & \omega_{\alpha,t} \end{matrix}$$

Now, each column could define a 2-degree polynomial. Specifically, if the prover $\mathcal{P}$ is honest, it will define $\frac{t}{2}$ polynomials for all α-wires as $f_1, \ldots, f_{\frac{t}{2}}$ and another $\frac{t}{2}$ polynomials for all β-wires as $g_1, \ldots, g_{\frac{t}{2}}$. Consider the following crucial equation:

$$\sum_{i\in[\frac{t}{2}]}\sum_{j\in[2]} f_i(j)\cdot g_i(j) = \sum_{i\in[t]}(\omega_{\alpha,i}\cdot\omega_{\beta,i}) = \sum_{i\in[t]}\omega_{\gamma,i}$$

Let's generalize the product polynomial as $h = \sum_{i\in[\frac{t}{2}]} f_i \cdot g_i \in \mathbb{F}[X]$. If all the multiplication triples are correct, the aggregated output must lie on the polynomial h. At this point, all parties can jointly verify whether $\sum_{j\in[2]}[\![h(j)]\!]_2 - [\![z]\!]_2 = [\![0]\!]_2$ by invoking the $\mathsf{CheckZero}$ subroutine. To complete the proof, the prover $\mathcal{P}$ must also demonstrate that the commitment to the polynomial h corresponds precisely to the inner product of $f_1, \ldots, f_{\frac{t}{2}}$ and $g_1, \ldots, g_{\frac{t}{2}}$. The key insight is that all parties can leverage the IT-MACs $\{[\![\omega_{\alpha,i}]\!]_2, [\![\omega_{\beta,i}]\!]_2\}_{i\in[t]}$ to homomorphically derive authenticated sharings of these $\frac{t}{2}$ polynomials, i.e., $[\![f_1]\!]_2, \ldots, [\![f_{\frac{t}{2}}]\!]_2$ and $[\![g_1]\!]_2, \ldots, [\![g_{\frac{t}{2}}]\!]_2$. A further verification of these commitments is performed by checking that $\sum_{i\in[\frac{t}{2}]}[\![f_i]\!]_2 \cdot [\![g_i]\!]_2 - [\![h]\!]_2 = [\![\tilde{0}]\!]_2$, where $\tilde{0}$ denotes a zero polynomial that *always* evaluates to 0. By the Schwartz-Zippel lemma, this can be verified by checking

$$\sum_{i\in[\frac{t}{2}]}[\![f_i(\eta)]\!]_2 \cdot [\![g_i(\eta)]\!]_2 - [\![h(\eta)]\!]_2 = [\![0]\!]_2$$

for a random $\eta \in \mathbb{F}_{2^\kappa}$. To ensure that this procedure is non-interactive, stateless, and deterministic, we generate η using the Fiat-Shamir heuristic, i.e., by hashing all transcripts.

Note that verifying $[\![f_1(\eta)]\!]_2, \ldots, [\![f_{\frac{t}{2}}(\eta)]\!]_2$, $[\![g_1(\eta)]\!]_2, \ldots, [\![g_{\frac{t}{2}}(\eta)]\!]_2$, and $[\![h(\eta)]\!]_2$ evaluated at the public value η reduces to another $\frac{t}{2}$-batched verification of multiplication triples. The parties can recursively perform this process until only one or two triples remain. The final multiplication triples can then be efficiently verified using the sacrifice technique [KOS16]. If any check fails, the party outputs false. Otherwise, all parties obtain a correctly authenticated vector $[\![\boldsymbol{r}]\!]_2$, which represents the secure evaluation output of $\mathsf{PRF}_{[\![\mathsf{dk}]\!]_2}(\mathsf{msg})$.

We then convert IT-MACs over $\mathbb{F}$ into IT-MACs over the group $\mathbb{G}$ with prime order q. Previously, Smart *et al.*[STA19] and Kondi *et al.*[KOR23] observed that IT-MACs over a group can be applied such that, for a group element $R \in \mathbb{G}$ with $R = r \cdot G$, it is secretly shared using correlations $\{R_i, M_i\} i \in [n]$, where $R = \sum_{i\in[n]} R_i and \sum_{i\in[n]} M_i = \Lambda \cdot R$. Here, $\Lambda \in \mathbb{Z}_q$ is the same global key used in $[\![r]\!]_q$. We extend their approach by converting $[\![\boldsymbol{r}]\!]_2$ to $[\![R]\!]_q$. The core challenge lies in the fact that $\boldsymbol{r} \in \mathbb{F}^\ell$, which is secretly shared over $\mathbb{F}$, cannot be directly aggregated into $[\![r]\!]_q$ or $[\![R]\!]_q$, both of which are shared over $\mathbb{Z}_q$ or a group.

Building on prior work [BST21], we adopt the concept of extended doubly-authenticated bits (edaBits) [EGK+20]. Specifically, we transform the original edaBits from the MPC setting into a multi-verifier zero-knowledge (MVZK)-friendly form $\mathsf{mv\text{-}edaBits} := \{([\![\rho]\!]_q, [\![\boldsymbol{\rho}[1]]\!]_2, \ldots, [\![\boldsymbol{\rho}[\ell]]\!]_2)\}$, where the random value

$\rho \in \mathbb{Z}_q$ is secret-shared over the arithmetic field $\mathbb{Z}_q$ *in the multi-verifier setting.* Its binary representation (i.e., $\rho = \sum_{j=1}^{\ell} 2^{j-1}\boldsymbol{\rho}[j] \bmod q$) is also secret-shared over the boolean field $\mathbb{F}_2$ *within the same multi-verifier setting.*

An important observation is that we need ρs in both fields to keep identical. Prior work [EGK+20] applied heavy cut-and-choose technique. This method is uneconomic to design multi-party EdDSA as just one mv-edaBits is used for a signature. Therefore, we improve the check protocol using a "sacrificing" technique. In particular, parties generate two edaBits, with $([\![a]\!]_q, [\![\boldsymbol{a}]\!]_2)$ and $([\![\rho]\!]_q, [\![\boldsymbol{\rho}]\!]_2)$ respectively. Incorporated with an affine circuit $\mathcal{C}_{\mathsf{aff}}$, $\mathcal{P}$ and $\mathcal{V}$s can securely evaluate to $b = a + \chi \cdot \rho \mod q$ in clear, where $\chi \in \mathbb{Z}_q$ is an unpredictable random value.

An important observation is that the values of ρ must remain identical across both fields. Prior work [EGK+20] used a heavy cut-and-choose technique, which is inefficient for designing multi-party EdDSA since each signature requires only one mv-edaBits. To address this, we improve the verification protocol using a 'sacrificing' technique. Specifically, the parties generate two edaBits: $([\![a]\!]_q, [\![\boldsymbol{a}]\!]_2)$ and $([\![\rho]\!]_q, [\![\boldsymbol{\rho}]\!]_2)$. By incorporating an affine circuit $\mathcal{C}_{\mathsf{aff}}$, the prover $\mathcal{P}$ and verifiers $\mathcal{V}$s can securely compute $b = a + \chi \cdot \rho \bmod q$ in the clear, where $\chi \in \mathbb{Z}_q$ is an unpredictable random value.

Suppose a dishonest $\mathcal{P}$ uses inconsistent values for a and ρ. All parties can detect this by invoking $\mathsf{CheckZero}([\![a]\!]_q + \chi \cdot [\![\rho]\!]_q - b)$. This works because, if the equation holds, we get:

$$a + \chi \cdot \rho - (a + \chi \cdot \rho),$$

where the red a and ρ are aggregated from the binary representations of mv-edaBits, and the blue a and ρ are generated from the integer part of the mv-edaBits. If Λ is uniformly random and $\chi \in \mathbb{Z}_q$ is computationally unpredictable, the adversary $\mathcal{A}$'s advantage in forging inconsistent mv-edaBits is negligible, i.e., $\mathsf{negl}(\kappa)$.

After the consistency of mv-edaBits, they can correctly convert the vector $[\![\boldsymbol{r}]\!]_2$ into the IT-MACs over group by $[\![R]\!]_q = (c - [\![\rho]\!]_q) \cdot G$ where $c = r + \rho \bmod q$ is evaluated by an addition circuit $\mathcal{C}_{\mathsf{add}}$, with inputs of $[\![\boldsymbol{r}]\!]_2$ and $[\![\boldsymbol{\rho}]\!]_2$.

3.3 Multi-Party EdDSA Signing

We focus on the Ed25519 version of the EdDSA signature, with the Ed448 case being nearly identical, differing only in the choice of PRF and elliptic curve. The multi-party EdDSA signing process follows two phases. In the key generation phase, each party generates all necessary keys: the secret key sk_i, signing key s_i, random seed k^*, derived key dk_i, and global keys Δ^i and Λ^i. Additionally, the public key for the signature is easily constructed using the additive cyclic group.

Given the message msg, a core step is to derive mv-edaBits non-interactively, statelessly, and deterministically through PCF evaluation. The session identifier is set as sid = msg, and all necessary common randomness is generated via $\mathsf{H}(k^*, \mathsf{sid}, \mathsf{mvnd})$. The verifiable nonce derivation can then proceed as follows:

1. Each P_i acts as the prover $\mathcal{P}$ to prove its R_i, while all other parties P_j, where $j \in [n] \setminus \{i\}$, act as the N verifiers $\mathcal{V}$.

2. P_i aborts if it detects any false in the multi-verifier nonce derivation process. Otherwise, it computes $R = \sum_{i\in[n]} R_i$.

The remaining steps are straightforward. Each P_i computes $h = \mathsf{H}_{\mathsf{sig}}(R, \mathsf{pk}, \mathsf{msg})$ and $\sigma_i = r_i + h \cdot s_i \bmod q$. All parties then open $\sigma = \sum_{i=1}^{n} \sigma_i \bmod q$. Finally, each party verifies the signature by checking if $\mathsf{verify}(\mathsf{pk}, \mathsf{msg}, (R, \sigma)) = \mathsf{false}$. If the verification fails, the process is aborted; otherwise, the parties output (R, σ). As a result, the overall computation and communication costs are primarily influenced by the multi-verifier nonce derivation.

4 The Designed Multi-Party EdDSA Signing Protocol

Recall that we have defined multi-party IT-MACs over groups in Section 3.1. Thus, this section directly shows the detailed multi-verifier nonce derivation protocol and its application in multi-party EdDSA signatures.

4.1 Extended Doubly-Authenticated Bits for MVZK Proof

Extended doubly-authenticated bits ($\mathsf{mv\text{-}edaBits}$) for multi-verifier zero-knowledge proof are a key tool in this work, enabling the efficient conversion of $[\![\boldsymbol{r}]\!]_2$ into an authenticated sharing $[\![R]\!]_q$ over the group. $\mathsf{mv\text{-}edaBits}$ are defined as a tuple $([\![\rho]\!]_q, \{[\![\boldsymbol{\rho}[1]]\!]_2, \ldots, [\![\boldsymbol{\rho}[\ell]]\!]_2\})$ where the identical random value $\rho \in \mathbb{Z}_q$ is secret-shared in the arithmetic domain $\mathbb{Z}_q$, and its binary representation bits are secret-shared in the binary domain $\mathbb{F}_{2^\kappa}$, i.e., $\rho = \sum_{i=1}^{\ell} 2^{i-1} \cdot \boldsymbol{\rho}[i] \mod q$. We present the ideal functionality for $\mathsf{mv\text{-}edaBits}$ in Fig. 9.

Before introducing the achievement of $\mathcal{F}_{\mathsf{MV\text{-}edaBits}}$, we first present a core check subroutine. This subroutine operates in the multi-verifier setting and utilizes a polynomial-based batch verification technique. A macro for this procedure is defined in Fig. 10. The secure instantiation protocol, $\mathsf{CheckMuls}$, is detailed in the full version. It extends the $\mathsf{AssertMultVec}$ protocol from[BMRS21] to the multi-verifier setting. As a result, the security of $\mathsf{CheckMuls}$ is analogous to the single-verifier protocol, with the distinction that an adversary may influence which honest verifier aborts, while others continue to output results. This issue can be resolved by having all verifiers broadcast their results and check for consistency. Any inconsistency will be detected as long as at least one verifier is honest. Since the simulator can simulate the input round of any subprotocol as described, it remains secure even if the adversary behaves inconsistently. Moreover, the protocol $\prod_{\mathsf{MV\text{-}edaBits}}$ ensures consistency, meaning that any malicious behavior where a corrupted prover sends different values to different honest verifiers will be detected.

The prover $\mathcal{P}$ and N verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_N$ can generate faulty $\mathsf{mv\text{-}edaBits}$ by simply invoking the PCF macro. We need to ensure the consistency between the bits $\{\boldsymbol{\rho}[1], \ldots, \boldsymbol{\rho}[\ell]\}$ and the random value ρ shared across two fields. This can be achieved through the cut-and-choose verification phase from [EGK+20]. In this paper, however, we observe that these values can be more efficiently verified using

Functionality $\mathcal{F}_{\mathsf{MV\text{-}edaBits}}$

Let $\mathbb{C}$ be the set of corrupted parties. This functionality runs with two types of parties, i.e., multiple verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_N$ and the prover $\mathcal{P}$. We use the symbols $\{[\![\cdot]\!]_q, [\![\cdot]\!]_2\}$ to distinguish the authenticated fields in the $\mathbb{Z}_q$ and $\mathbb{F}_{2^\kappa}$ respectively.

Initialize: For each $\mathcal{V}_i, i \in [N]$, upon receiving (init, i) from $\mathcal{V}_i$ and $\mathcal{P}$, sample $\Delta^i \leftarrow \mathbb{F}_{2^\kappa}, \Lambda^i \leftarrow \mathbb{Z}_q$. If $\mathcal{V}_i$ is corrupted then receive $(\Delta^i \in \mathbb{F}_{2^\kappa}, \Lambda^i \in \mathbb{Z}_q)$ from the adversary. Here, Δ^i is global key for $[\![\cdot]\!]_2$-sharing and Λ^i is global key for $[\![\cdot]\!]_q$-sharing. Store (Δ^i, Λ^i) and send them to $\mathcal{V}_i$, and ignore all subsequent (init, i) commands.

Create edaBits: Upon receiving $(\mathsf{edaBits}, str)$ from $\mathcal{V}$s and $\mathcal{P}$, if (Δ^i, Λ^i) for $i \in [N]$ have been stored:

- If sid has never been received before, generates $([\![\rho]\!]_q, \{[\![\boldsymbol{\rho}[1]]\!]_2, \ldots, [\![\boldsymbol{\rho}[\ell]]\!]_2\})$ satisfying $\rho = \sum_{i\in[\ell]} 2^{i-1} \cdot \boldsymbol{\rho}[i] \mod q$, sends them to all parties and stores $(sid, ([\![\rho]\!]_q, \{[\![\boldsymbol{\rho}[1]]\!]_2, \ldots, [\![\boldsymbol{\rho}[\ell]]\!]_2\}))$;
- Otherwise, it finds the record as $(sid, *)$ and sends $([\![\rho]\!]_q, \{[\![\boldsymbol{\rho}[1]]\!]_2, \ldots, [\![\boldsymbol{\rho}[\ell]]\!]_2\})$ to all parties.

Fig. 9: The Ideal Functionality for mv-edaBits

Macro CheckMuls

This macro is executed with $\mathcal{P}$ and N verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_N$, and inherit all the features of PCF (shown in Fig. 7) and $\mathcal{F}_{\mathsf{MV\text{-}ND}}$ (shown in Fig. 12). Furthermore, this macro is invoked by the following commands.

Check Multiplications: Upon receiving t multiplication triples $\{([\![\omega_{\alpha,j}]\!]_2, [\![\omega_{\beta,j}]\!]_2, [\![\omega_{\gamma,j})]\!]_2\}_{j\in[t]}$ from $\mathcal{P}$ and $\mathcal{V}$s, where t multiplication tuples are equipped with IT-MACs. The details of this macro are provided in Appendix A and Fig. 16. Finally, if for any $j \in [t]$ s.t. $\omega_{\gamma,j} \neq \omega_{\alpha,j} \cdot \omega_{\beta,j}$, then set $res = \mathsf{false}$. Otherwise, set $res = \mathsf{true}$.

Fig. 10: The Multi-Verifier Check Multiplications Macro

the sacrifice of another set of mv-edaBits. As shown in Theorem 1, if the red ρ_i values (the Binary parts) are inconsistent with the blue ρ (the Arithmetic part), the check subroutine will fail, except with a probability of at most $\frac{1}{q} + \mathsf{negl}(\kappa)$.

Communication and Round Complexity. The transcripts sent from $\mathcal{P}$ to $\mathcal{V}$s are mainly caused during the gate-by-gate paradigm and polynomial-based batch verification process. The former consumes t_2 bits, where t_2 is the number of AND gates in $\mathcal{C}_{\mathsf{aff}}$; the latter consumes $\log(t_2) \cdot 4\kappa + 9\kappa$ bits. Thus, the communication complexity of $\prod_{\mathsf{MV\text{-}edaBits}}$ is roughly $O(t_2 + \log(t_2) \cdot \kappa)$. In terms of round complexity, since the affine circuit $\mathcal{C}_{\mathsf{aff}}$ is publicly known, all transcripts for the

Protocol $\prod_{\mathsf{MV\text{-}edaBits}}$

This protocol runs with two types of parties, i.e., multiple verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_N$ and the prover $\mathcal{P}$. We use the symbols $\{[\![\cdot]\!]_q, [\![\cdot]\!]_2\}$ to distinguish the authenticated fields in the $\mathbb{Z}_q$ and $\mathbb{F}_{2^\kappa}$ respectively. Parameterized by the security parameter κ. All parties hold an affine circuit $\mathcal{C}_{\mathsf{aff}} : \{x + \chi \cdot y = z\}$, where x, χ, y are ℓ-bit inputs, and z is an ℓ-bit output. The number of multiplication gates in $\mathcal{C}_{\mathsf{aff}}$ is denoted as $t_2 = |\mathcal{C}_{\mathsf{aff}}|$. Let $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_q$.

Setup runs $\mathsf{PCF.Gen}_{\mathsf{mv}}$ to generates keys for each $\mathcal{V}_i, i \in [N]$ and $\mathcal{P}$.

1. $\mathcal{P}$ and $\mathcal{V}$s run $\mathsf{PCF.Gen}_{\mathsf{mv}}(1^\kappa, \mathbb{Z}_q)$ to generate $(k_0^{(q)})$ for $\mathcal{P}$ and $(k_i^{(q)})$ for $\mathcal{V}_i$.
2. $\mathcal{P}$ and $\mathcal{V}$s run $\mathsf{PCF.Gen}_{\mathsf{mv}}(1^\kappa, \mathbb{F}_2)$ to generate $(k_0^{(2)})$ for $\mathcal{P}$ and $(k_i^{(2)})$ for $\mathcal{V}_i$.
3. Parties jointly sample common random source $k^* \leftarrow \mathbb{F}_{2^\kappa}$.

Create mv-edaBits runs $\mathsf{PCF.Eval}_{\mathsf{mv}}$ to derive authenticated mv-edaBits for each $\mathcal{V}_i, i \in [N]$ and $\mathcal{P}$. Besides, Create needs an input of a common string str. All parties execute as follows:

1. $\mathcal{P}$ runs $\mathsf{PCF.Eval}_{\mathsf{mv}}(0, k_0^{(q)}, \mathsf{H}(str||1))$, while for each $i \in [N]$, $\mathcal{V}_i$ runs $\mathsf{PCF.Eval}_{\mathsf{mv}}(i, k_i^{(q)}, \mathsf{H}(str||1))$ to generate two $[\![\rho]\!]_q, [\![a]\!]_q$.
2. In parallel, $\mathcal{P}$ runs $\mathsf{PCF.Eval}_{\mathsf{mv}}(0, k_0^{(2)}, \mathsf{H}(str||2))$ and for each $i \in [N]$, $\mathcal{V}_i$ runs $\mathsf{PCF.Eval}_{\mathsf{mv}}(i, k_i^{(2)}, \mathsf{H}(str||2))$ to generate $[\![\boldsymbol{\rho}]\!]_2, [\![\boldsymbol{a}]\!]_2$.
3. Each party generates a common random by $\mathsf{H}(str||\mathsf{H}(str||1)||\mathsf{H}(str||2)) = \chi \in \mathbb{Z}_q$ and append $str := str||\mathsf{H}(str||1)||\mathsf{H}(str||2)||\chi$.
4. For circuit $\mathcal{C}_{\mathsf{aff}}$, they evaluate it with the inputs of $[\![\boldsymbol{a}]\!]_2$, χ and $[\![\boldsymbol{\rho}]\!]_2$. The gate-by-gate circuit evaluations are executed as follows:
   (a) In a topological order, for each gate $(\alpha, \beta, \gamma, T) \in \mathcal{C}_{\mathsf{aff}}$ with input wire values of $(\omega_\alpha, \omega_\beta)$ and output wire value of ω_γ:
   - If $T = \mathsf{ADD}$, $\mathcal{P}$ and $\mathcal{V}$s locally compute $[\![\omega_\gamma]\!]_2 = [\![\omega_\alpha]\!]_2 + [\![\omega_\beta]\!]_2$.
   - If $T = \mathsf{MULT}$ and this is j-th multiplication gate, $\mathcal{P}$ and $\mathcal{V}$s execute $[\![\omega_\gamma]\!]_2 \leftarrow \mathsf{Assign}(\omega_\alpha \cdot \omega_\beta)$.
   - Append $str := str||d_j$ where d_j is the transcript sent by $\mathcal{P}$.

   (b) $\mathcal{P}$ and $\mathcal{V}$s jointly check the correctness of multiplication triples by invoking
   $$\mathsf{CheckMuls}(\{[\![\omega_{\alpha,j}]\!]_2, [\![\omega_{\beta,j}]\!]_2, [\![\omega_{\gamma,j}]\!]_2\}_{j \in [t_2]})$$
   with seed $str := \mathsf{H}(str)$. If any failure happens, the parties output false.
5. Parties now obtain the output wires $[\![\boldsymbol{b}]\!]_2$ that satisfy $\sum_{j=1}^{\ell} 2^{j-1} \cdot \boldsymbol{b}[j] = (\sum_{j=1}^{\ell} 2^{j-1} \cdot \boldsymbol{a}[j]) + \chi \cdot (\sum_{j=1}^{\ell} 2^{j-1} \cdot \boldsymbol{\rho}[j]) \mod q$.
6. Parties open $\boldsymbol{b}$. Then, all parties invoke $\mathsf{CheckZero}([\![a]\!]_q + \chi \cdot [\![\rho]\!]_q - b)$.
7. If any checking fails, the parties output false and abort. Otherwise, they output $([\![\rho]\!]_q, \{[\![\boldsymbol{\rho}[1]]\!]_2, \ldots, [\![\boldsymbol{\rho}[\ell]]\!]_2\})$.

Fig. 11: The Generation Protocol of mv-edaBits

AND gates can be sent in a single round. Similarly, for the polynomial-based batch verification process, $\mathcal{P}$ can transmit all verification proofs in one round

Functionality $\mathcal{F}_{\mathsf{MV\text{-}ND}}$

This functionality runs with $\mathcal{P}$ and $\mathcal{V}_1, \ldots, \mathcal{V}_N$. This functionality is parameterized by the nonce derivation circuit $\mathcal{C}^* := \{R = \mathsf{PRF}_{\mathsf{dk}}(\mathsf{msg}) \cdot G\}$ where G is a generator of the elliptic curve group $\mathbb{G}$ with order q.

1. Upon receiving $(\mathsf{mvzk\text{-}input}, did, \mathsf{dk})$ from $\mathcal{P}$ and $(\mathsf{mvzk\text{-}input}, i)$ from all the verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_N$, with a fresh identifier sid, store (did, dk).
2. Upon receiving $(\mathsf{mvzk\text{-}prove}, did, \mathsf{msg})$ from $\mathcal{P}$ and $(\mathsf{mvzk\text{-}verify}, did, \mathsf{msg})$ from $\mathcal{V}_1, \ldots, \mathcal{V}_N$. If (did, dk) has been stored, set $res = \mathsf{true}, R := \mathcal{C}^*(\mathsf{dk}, \mathsf{msg})$ and $res = \mathsf{false}$ otherwise.
3. If $res = \mathsf{true}$, send $(\mathsf{mvzk\text{-}proof}, \mathsf{msg}, [\![R]\!]_q)$ to the parties; otherwise, send abort to the parties.

Fig. 12: Multi-Verifier Zero-Knowledge Proof Functionality for Nonce Derivation

using the Fiat-Shamir heuristic. As a result, this protocol can be executed in a *non-interactive* setting.

Theorem 1. $\prod_{\mathsf{MV\text{-}edaBits}}$ *UC-realizes* $\mathcal{F}_{\mathsf{MV\text{-}edaBits}}$ *in the presence of an adversary statically corrupting up to* $n-1$ *parties, in the* $(\mathcal{F}_{\mathsf{MV\text{-}CheckMULs}})$*-hybrid random oracle model and* PCF *assumption.*

The security proof could be found in the full version.

4.2 Multi-Verifier Nonce Derivation

Our multi-verifier zero-knowledge proof (MVZK)-based nonce derivation protocol follows a gate-by-gate paradigm, where the value on each wire is formally secretly shared as $[\![x]\!]_2 = \{(x, m_1, \ldots, m_N), k_1, \ldots, k_N\}$ for N verifiers, such that $m_i = k_i + x \cdot \Delta^i \in \mathbb{F}_{2^\kappa}$, and each $\mathcal{V}_i$ holds $(k_i, \Delta^i), i \in [N]$. It could be generated by invoking PCF (see Fig. 7). $\mathcal{F}_{\mathsf{MV\text{-}ND}}$ shown in Fig. 12 defines our multi-verifier nonce derivation functionality. Our multi-verifier zero-knowledge proof (MVZK)-based nonce derivation protocol follows a gate-by-gate paradigm, where the value on each wire is secret-shared as $[\![x]\!]_2 = \{(x, m_1, \ldots, m_N), k_1, \ldots, k_N\}$ for N verifiers. Here, $m_i = k_i + x \cdot \Delta^i \in \mathbb{F}_{2^\kappa}$, and each $\mathcal{V}_i$ holds (k_i, Δ^i) for $i \in [N]$. This can be generated by invoking PCF (see Fig.7). Our multi-verifier nonce derivation functionality is formally defined by $\mathcal{F}_{\mathsf{MV\text{-}ND}}$ in Fig.12.

Given the $\mathcal{F}_{\mathsf{MV\text{-}CheckMULs}}$ and $\mathcal{F}_{\mathsf{MV\text{-}edaBits}}$ ideal functionalities, we design an instance protocol for multi-verifier zero-knowledge (MVZK) proof of nonce derivation within the $(\mathcal{F}_{\mathsf{MV\text{-}CheckMULs}}, \mathcal{F}_{\mathsf{MV\text{-}edaBits}})$-hybrid model and under the PCF assumption, as illustrated in Fig. 13. The $\prod_{\mathsf{MV\text{-}ND}}$ phase is designed to be *deterministic* and *stateless* to support multi-party EdDSA signing, where the common random generators are instantiated using $\mathsf{H}(\mathsf{long\text{-}term\ key} || \mathsf{transcripts})$ with a cryptographic hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^*$.

Protocol $\prod_{\mathsf{MV\text{-}ND}}$

Parameterized by the security parameter κ, an elliptic curve group $\mathbb{G}$ generated by G with order q. n parties hold a circuit for a keyed PRF function $\mathcal{C} := \mathsf{PRF}_{\mathsf{dk}}(\mathsf{msg})$. The circuit $\mathcal{C}$ consists of t multiplication gates, with inputs including the secret shared derived key $\mathsf{dk} \in \mathbb{F}_{2^\kappa}$ and the message $\mathsf{msg} \in \{0,1\}^\ell$, and it produces a secretly-shared nonce $\boldsymbol{r} \in \{0,1\}^\ell$ as the output. Furthermore, all parties consensus on an addition circuit $\mathcal{C}_{\mathsf{add}} : \{x + y = z\}$ with $t' = |\mathcal{C}_{\mathsf{add}}|$ multiplication gates. *Let $\tilde{\mathcal{C}}$ being a circuit computed via the $\mathcal{C}$ followed by $\mathcal{C}_{\mathsf{add}}$ and $\tilde{t} = t + t'$.* When each party acts as prover $\mathcal{P}$ and the other $N = n - 1$ parties act as verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_N$ in turns.

Setup: Run once, with $\mathcal{P}$ and $\mathcal{V}$s invoke the Setup phase of $\mathcal{F}_{\mathsf{MV\text{-}edaBits}}$ (in Fig. 9). Furthermore, it receives $[\![\mathsf{dk}]\!]_2$ from $\mathcal{P}$.

Proof: Each party inputs publicly known message msg:

1. Each party initializes a string as $str := \mathsf{H}(\mathsf{mvnd}||\mathsf{msg}||k^*)$.
2. All parties generates $\mathsf{mv\text{-}edaBits}$ by sending $(\mathsf{edaBits}, str)$ to $\mathcal{F}_{\mathsf{MV\text{-}edaBits}}$.

// Gate-by-Gate Evaluation of $\tilde{\mathcal{C}}$

3. For circuit PRF, they evaluate it with the inputs of $[\![\mathsf{dk}]\!]_2$ and msg, while for circuit $\mathcal{C}_{\mathsf{add}}$, they evaluate it with the inputs of $[\![\boldsymbol{r}]\!]_2$ and $[\![\boldsymbol{\rho}]\!]_2$. Initializing str as msg. The gate-by-gate circuit evaluations are executed as follows:
   (a) In a topological order, for each gate $(\alpha, \beta, \gamma, T) \in \tilde{\mathcal{C}}$ with input wire values of $(\omega_\alpha, \omega_\beta)$ and output wire value of ω_γ:
   - If $T = \mathsf{ADD}$, $\mathcal{P}$ and $\mathcal{V}$s locally compute $[\![\omega_\gamma]\!]_2 = [\![\omega_\alpha]\!]_2 + [\![\omega_\beta]\!]_2$.
   - If $T = \mathsf{MULT}$ and this is j-th multiplication gate, $\mathcal{P}$ and $\mathcal{V}$s execute $[\![\omega_\gamma]\!]_2 \leftarrow \mathsf{Assign}(\omega_\alpha \cdot \omega_\beta)$. (Append $str := str||d_j$ where d_j is the transcript sent by $\mathcal{P}$)

   (b) $\mathcal{P}$ and $\mathcal{V}$s jointly check the correctness of multiplication triples by invoking

   $$\mathsf{CheckMuls}(\{[\![\omega_{\alpha,j}]\!]_2, [\![\omega_{\beta,j}]\!]_2, [\![\omega_{\gamma,j}]\!]_2\}_{j \in [\tilde{t}]}).$$

   with seed $str := \mathsf{H}(str)$. If any failure happens, the parties output false. Otherwise, they obtain $[\![\boldsymbol{c}]\!]_2$ satisfying $\sum_{j\in[\ell]} 2^{j-1} \cdot \boldsymbol{c}[j] = \sum_{j\in[\ell]} 2^{j-1} \cdot \boldsymbol{r}[j] + \sum_{j\in[\ell]} 2^{j-1} \cdot \boldsymbol{\rho}[j] \mod q$ and $\boldsymbol{r} = \mathsf{PRF}_{\mathsf{dk}}(\mathsf{msg})$.

// Conversion between $\mathbb{F}_{2^\kappa}$ field and $\mathbb{G}$ group

4. All parties opens $\boldsymbol{c}$ and computes $c = \sum_{j\in[\ell]} 2^{j-1} \cdot \boldsymbol{c}[j]$.
5. If any failure happens, the parties output false. Otherwise, each party outputs the authenticated nonce as $[\![R]\!]_q = (c - [\![\rho]\!]_q) \cdot G$ and $[\![r]\!]_q = c - [\![\rho]\!]_q$.

Fig. 13: Multi-Verifier Stateless Deterministic Nonce Derivation based on MVZK

Communication and Round Complexity. The communication complexity consists of three parts: (1) the gate-by-gate paradigm requires $\tilde{t}$ bits, where $\tilde{t}$ is the number of multiplication gates in the combined circuit $\tilde{\mathcal{C}}$, resulting one round communication; (2) the polynomial-based batch verification technique

consumes $\log(\tilde{t}) \cdot 4\kappa + 9\kappa$ bits, resulting in a total communication complexity of approximately $O(\tilde{t} + \log(\tilde{t}) \cdot \kappa)$ in a *non-interactive* setting; (3) the generation of $\mathsf{mv\text{-}edaBits}$ requires $t_2 + \log(t_2) \cdot 4\kappa + 9\kappa + \ell + 2\ell \cdot \kappa$ bits per party, with one rounds of communication. Considering the sizes of the PRF and the affine circuit, we have $\tilde{t} \gg t_2$. The three components mentioned above are communication-independent, meaning they can all be merged into a single round. Therefore, the communication complexity of the $\prod_{\mathsf{MV\text{-}ND}}$ protocol is $O(n(\tilde{t} + \log(\tilde{t}) \cdot \kappa + \kappa \cdot \ell))$ bits per party, with only one round of communication in total.

Here, $\mathcal{P}$ only needs to send d_i to all $\mathcal{V}$s. With a binary tree pattern [QYYZ22], the amortized communication complexity can be optimized from $O(n)$ to $O(1)$ among the n signing parties, at the cost of increasing the rounds from $O(1)$ to $O(n)$. Alternatively, a parent node forwards the same d_i to its *three or more* child nodes and this process iterates until the edge nodes are reached, providing a trade-off between communication and rounds. Compared to the protocol by Garillot et al. [GKMN21], where each $\mathcal{V}_i$ for $i \in [N]$ sends an independent garbled circuit to $\mathcal{P}$, our $\mathcal{F}_{\mathsf{MV\text{-}ND}}$ protocol is better suited for large-scale ZK implementations in high-speed networks, such as LANs.

We prove that the multi-verifier stateless deterministic nonce derivation protocol is UC-secure, even when an adversary statically corrupts up to $n-1$ parties.

Theorem 2. $\prod_{\mathsf{MV\text{-}ND}}$ *UC-realizes* $\mathcal{F}_{\mathsf{MV\text{-}ND}}$ *in the presence of an adversary statically corrupting up to* $n-1$ *parties, in the* $(\mathcal{F}_{\mathsf{MV\text{-}CheckMULs}}, \mathcal{F}_{\mathsf{MV\text{-}edaBits}})$*-hybrid random oracle model and* PCF *assumption.*

The security proof can be found in the full version.

4.3 Multi-Party EdDSA Signature Protocol

Building on prior definitions [LN18,BST21], we present the ideal functionality for EdDSA, as illustrated in Fig. 14. In $\mathcal{F}_{\mathsf{EdDSA}}$, the (`KeyGen`) command can be called *only once*, while the (`Sign`) command can be invoked multiple times.

In Fig. 15, we present the details of the multi-party EdDSA signing protocol, which allows us to achieve $O(n)$ communication bandwidth. This protocol operates within the $(\mathcal{F}_{\mathsf{MV\text{-}ND}}, \mathcal{F}_{\mathsf{Com}}, \mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{DL}}})$-hybrid model and is executed by n parties. During the key generation phase, each party P_i, where $i \in [n]$, generates its secret key sk_i and jointly computes the public key as $\mathsf{pk} = \sum_{i \in [n]} s_i \cdot G$. Party P_i also invokes $\mathcal{F}_{\mathsf{MV\text{-}ND}}$ to commit to the PRF key $\mathsf{dk}_i := \mathsf{PRF}(\mathsf{sk}_i)[\ell_b + 1; 2\ell_b]$, where P_i acts as the prover and the other $n-1$ parties serve as verifiers ($N = n-1$). It is important to note that at this stage, each party P_i generates all necessary uniformly random long-term keys, including k^*, Δ^i, Λ^i, and PCF keys for $\prod_{\mathsf{MV\text{-}ND}}$. The PCF can be implemented using a client-server model, where key management servers hold the client's keys in escrow, and the client runs $\mathsf{PCF.Gen}_{\mathsf{mv}}$ on behalf of the servers. This approach is becoming a popular service in lightweight internet environments due to its features of availability and scalability. Benefiting from the stateless and deterministic design, our protocol offers a promising solution, as it eliminates the need for key management servers to handle state

Functionality $\mathcal{F}_{\mathsf{EdDSA}}$

This functionality is parameterized by a set of EdDSA parameters, i.e., $\mathtt{params} = (\mathbb{E}_p, \mathbb{G}, q, G, \ell_b, \ell, \mathsf{H}_{\mathsf{sig}}, \mathsf{PRF})$ and runs with parties $P_1, \ldots, P_n$ as follows:

- Upon receiving $(\mathtt{KeyGen}, \mathtt{params})$ from all parties, generate a key pair $(\mathsf{dk}, s, \mathsf{pk})$ by running $\mathbf{KeyGen}(\mathtt{params})$, and store $(\mathsf{pk}, \mathsf{dk}, s)$. Then, send pk to $P_1, \ldots, P_n$, and ignore all subsequent $(\mathtt{KeyGen})$ commands.
- Upon receiving $(\mathtt{Sign}, \mathsf{msg})$ from all parties, if $(\mathtt{KeyGen})$ has not been called then abort; if msg has been signed previously, then send $(\mathsf{msg}, (\sigma, R))$ back; otherwise, generate an EdDSA signature (σ, R) by running $\mathbf{Sign}(\mathsf{dk}, s, \mathsf{pk}, \mathsf{msg})$ and output $(\mathsf{msg}, (\sigma, R))$ to all parties.

Fig. 14: The EdDSA Functionality

synchronization or rely on high-entropy random number generators, thereby reducing deployment costs.

During the signing phase, each party verifiably generates $[\![R_i]\!]_q$ that equals to $\mathsf{PRF}_{[\![\mathsf{dk}_i]\!]_2}(\mathsf{msg}) \cdot G$ by invoking $\mathcal{F}_{\mathsf{MV\text{-}ND}}$. If all parties receive $res \neq \perp$ from $\mathcal{F}_{\mathsf{MV\text{-}ND}}$, they obtain an authenticated $[\![R_i]\!]_q$. After opening and verifying $R_1, \ldots, R_n$, all parties can correctly compute $R = \sum_{i \in [n]} R_i$. The communication overheads required is $|\mathcal{F}_{\mathsf{MV\text{-}ND}}| + 2 * \ell_{\mathbb{G}} + q$ for each party. As discussed in Section 4.2, we instantiate $\mathcal{F}_{\mathsf{MV\text{-}ND}}$ in a single round using the Fiat-Shamir heuristic; therefore, the communication rounds required for the signing phase total three.

Theorem 3. *Assume that* PRF *is a pseudorandom function. Then,* $\prod_{\mathsf{MP,Sign}}$ *UC-realizes* $\mathcal{F}_{\mathsf{EdDSA}}$ *in the presence of an adversary statically corrupting up to* $n-1$ *parties, in the* $(\mathcal{F}_{\mathsf{MV\text{-}ND}}, \mathcal{F}_{\mathsf{Com}}, \mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{DL}}})$*-hybrid random oracle model.*

The security proof can be found in the full version.

Extension to Threshold Case. Our multi-party EdDSA protocol can be extended to the generic-threshold setting, where any $t+1$-out-of-n parties can generate a signature, and at most t parties can be corrupted This extension adopts the approach in prior works, e.g., [GG18,DKLs18,LN18,DKLs19,DKLS24] using Feldman's VSS and Shamir secret sharing (SSS). Specifically, the randomized key-generation protocol involves: (1) generating the public key: all parties securely compute public/private keys following the approach in prior works; (2) producing short PCF keys: either all parties run a threshold MPC protocol to generate PCF keys, or a client generates and distributes them. For deterministic signing protocol, any $t+1$ parties can locally convert the SSS-based key shares into an additively-shared piece and finish the signing. This conversion is standard and highly efficient. One caveat is that signatures produced by different sets of $t+1$ parties on the same message differ yet can be verified with the same public key. However, this still satisfies the deterministic requirement when considering the fixed set of $t+1$ parties.

Protocol $\prod_{\mathsf{MP,Sign}}$

This protocol is run among multiple parties $P_1, \ldots, P_n$ and is parameterized by the EdDSA parameters $\mathtt{params} = (\mathbb{E}_p, \mathbb{G}, q, G, \ell_b, \ell, \mathsf{H_{sig}}, \mathsf{PRF})$, with $\ell = 2\ell_b$, $\mathsf{H_{sig}}$ is hash function for signature and PRF is instantiated by $\mathtt{SHA512}$. This protocol makes use of the ideal oracle $\mathcal{F}_{\mathsf{MV\text{-}ND}}$ (Fig. 12).

Distributed Key Generation: Upon receiving $(\mathsf{KeyGen}, \mathtt{params})$, each party $P_i, i \in [n]$ executes as follows:

1. P_i samples private key as $\mathsf{sk}_i \leftarrow \{0,1\}^{\ell_b}$ and computes $(\boldsymbol{h}_i[1], \ldots, \boldsymbol{h}_i[\ell]) := \mathsf{PRF}(\mathsf{sk}_i)$.
2. P_i sets derived key as $\mathsf{dk}_i := \{\boldsymbol{h}_i[\ell_b + 1], \ldots, \boldsymbol{h}_i[2\ell_b])\}$, sends $(\mathsf{mvzk\text{-}input}, did_i, \mathsf{dk}_i)$ to $\mathcal{F}_{\mathsf{MV\text{-}ND}}$ with constant identifier $\mathsf{did_i}$.
3. P_i sets $\boldsymbol{h}_i[1] = \boldsymbol{h}_i[2] = \boldsymbol{h}_i[3] = \boldsymbol{h}_i[\ell_b] := 0$ and $\boldsymbol{h}_i[\ell_b - 1] := 1$, then use the updated vector $(\boldsymbol{h}_i[1], \ldots, \boldsymbol{h}_i[\ell_b])$ to define $s_i = \sum_{j=1}^{\ell_b} 2^{j-1} \cdot \boldsymbol{h}_i[j] \mod q$.
4. P_i computes public key share as $\mathsf{pk}_i = s_i \cdot G$.
5. All parties send pk_i for $i \in [n]$ using $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{DL}}}$.
6. After receiving correct pk_i for all $i \in [n]$ from $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{DL}}}$, P_i computes common public key $\mathsf{pk} = \sum_{i \in [n]} \mathsf{pk}_i$ and stores $\{\mathsf{pk}, \mathsf{sk}_i, \mathsf{dk}_i, s_i\}$.
7. The key generation phase is run only once.

Distributed Signing: With common input $(\mathsf{Sign}, \mathsf{msg})$, each party $P_i, i \in [n]$ executes as follows:

1. Each party P_i acts as $\mathcal{P}$ by sending $(\mathsf{mvzk\text{-}prove}, did_i, \mathsf{msg})$ to $\mathcal{F}_{\mathsf{MV\text{-}ND}}$ while all other P_j, $j \neq i$, send $(\mathsf{mvzk\text{-}verify}, did_i, \mathsf{msg})$ to $\mathcal{F}_{\mathsf{MV\text{-}ND}}$ acting as $\mathcal{V}$s with $N = n - 1$.
2. Upon receiving $[\![R_1]\!]_q, \ldots, [\![R_n]\!]_q$ and r_i from $\mathcal{F}_{\mathsf{MV\text{-}ND}}$, each P_i computes $[\![R]\!]_q = \sum_{i \in [n]} [\![R_i]\!]_q$. All parties jointly open to R. P_i aborts if any $res_j = \mathsf{abort}$ for $j \neq i$ or R is incorrectly opened.
3. P_i locally computes $h = \mathsf{H_{sig}}(\mathsf{pk}, R, \mathsf{msg})$ and the signature share $\sigma_i = s_i \cdot h + r_i \mod q$. Then P_i sends σ_i to all the parties using commitment $\mathcal{F}_{\mathsf{Com}}$.
4. Upon receiving all the $\sigma_j, j \neq i$ from $\mathcal{F}_{\mathsf{Com}}$, each party computes $\sigma = \sum_{i \in [n]} \sigma_i \mod q$. If (σ, R) is not a valid signature on msg, then P_i aborts. Otherwise, P_i outputs (σ, R).

Fig. 15: The Stateless Deterministic Multi-Party EdDSA Signing Protocol

5 Performance and Evaluation

The evaluation is configured using the Ed25519 curve, providing a security level of $\kappa = 128$ and utilizing SHA512 as the PRF nonce derivation function, as specified by the EdDSA standard. According to the estimates in [AAL+24], SHA512 involves approximately $58k$ AND gates. The multi-party signing protocol is essentially a lightweight wrapper around $\mathcal{F}_{\mathsf{MV\text{-}ND}}$, with the primary cost driven by executing $\mathcal{F}_{\mathsf{MV\text{-}ND}}$ among the parties. Notably, the structure of our signing

protocol, $\prod_{\mathsf{MV\text{-}ND}}$, ensures minimal computational overhead when instantiating $\mathcal{F}_{\mathsf{MV\text{-}ND}}$ in both directions. While $\mathcal{P}$ evaluates the circuit, the $\mathcal{V}$s remain idle, and when the $\mathcal{V}$s validate the circuit,$\mathcal{P}$ has no additional tasks. As a result, when a party P_i acts as $\mathcal{P}$ during its nonce verification session, it idles in its $\mathcal{V}_j$ role for $j \in [N]$ during the other parties' nonce verification sessions. Consequently, the workload is evenly distributed across all parties.

- Gate-by-Gate Evaluation. As discussed in Section 4.2, each AND gate requires one VOLE instance for a single transfer and secret addition. Therefore, each party primarily needs approximately $58k$ VOLEs for the gate-by-gate evaluation phase. The evaluations of affine circuit $\mathcal{C}_{\mathsf{aff}}$ during the generation of mv-edaBits and addition circuit $\mathcal{C}_{\mathsf{add}}$ after the PRF circuit introduce minimal additional VOLE overhead.
- CheckMuls. For EdDSA, using the Fiat-Shamir heuristic, the parties need to hash $50.9KB$ of messages, followed by $\log(58k)$ computational iterations within a single transfer. Each iteration involves hashing $112B$ messages, performing one polynomial inter-product over $\mathbb{F}_{2^\kappa}[X]$ of degree 2, executing 3κ VOLEs, and conducting $2t$ polynomial evaluations of degree 1. The final iteration requires four VOLE correlations and hashing $128B$ messages. Thus, each party needs approximately $3\kappa \times \log(58k) + 4 \approx 6.1k$ VOLEs, with only minor overhead.
- Consistency Check. Each party additionally generates up to two mv-edaBits, consuming around $2|q| + 2\ell$ VOLEs, and performs n elliptic curve multiplications and additions with negligible overhead.

In summary, the workload for each party is approximately $65.6k$ VOLEs, with minimal additional overhead. Boyle et al. [BCG+22] provide estimates for PCF evaluation times. Specifically, VOLEs can be instantiated using fixed-key AES, measured by AES-NI instructions on modern CPUs. On a 3GHz processor, a single PCF evaluation takes approximately 3.57×10^{-3} milliseconds. Based on this, we estimate that generating one signature between two parties with one corruption takes around 230 ms. This process can be scaled linearly using GPUs, as the AES calls in PCF are fully independent and thus parallelizable. However, the lack of publicly available open-source code for PCF [BCG+22] presents a significant challenge to our evaluation, meaning the timing estimates provided here are purely theoretical.

We compare our work with two existing multi-party EdDSA signing protocols: Bonte et al. [BST21], which operates in the honest majority setting, and Garillot et al. [GKMN21], which targets the dishonest majority setting. Bonte et al. [BST21] conducted their experiments using SCALE-MAMBA in a LAN environment, with each party running on an Intel i7-7700K CPU (4 cores at 4.2GHz, 2 threads per core) and 32GB of RAM, connected via a 10Gb/s network switch. The protocol achieved an average runtime of 1406 ms under the Shamir (3,1) access structure, based on 100 trials. The total computational burden for each party in Garillot et al. [GKMN21] involves approximately 132k AES invocations on 128-bit ciphertexts, hashing a 245KB message, performing three elliptic curve

multiplications, and executing 256 additions in $\mathbb{Z}_q$. However, Garillot et al. did not provide empirical measurements for their $\pi_{n,\mathsf{Sign}}$ protocol. To offer a peer-to-peer comparison, we estimate the performance of Garillot et al. 's protocol using the benchmarks from Boyle et al. [BCG+22], where each byte of fixed-key AES requires approximately 1.3 CPU cycles. On a 3GHz processor, this results in an estimated signature time of 102 *ms*. Although our computation time is relatively modest, we achieve one to two orders of magnitude improvement in communication efficiency compared to prior works.

Acknowledgments. Qi Feng is supported by the National Key Research and Development Program of China (Grant No. 2021YFA1000600), the National Natural Science Foundation of China (Grant Nos. 62202339, 62172307, U21A20466), the Major Program (JD) of Hubei Province (No. 2023BAA027) and the Science and Technology on Communication Security Laboratory Foundation (Grant No. 6142103022202). Kang Yang is supported by the National Natural Science Foundation of China (Grant Nos. 62102037). Yu Yu is supported by the National Natural Science Foundation of China (Grant Nos. 92270201 and 62125204). Yu Yu's work has also been supported by the New Corner-stone Science Foundation through the XPLORER PRIZE.

References

AAL+24. David Archer, Victor Arribas Abril, Steve Lu, Pieter Maene, Nele Mertens, Danilo Sijacic, and Nigel Smart. 'Bristol Fashion' MPC Circuits. https://nigelsmart.github.io/MPC-Circuits/, Accessed at Jan 2024.

BBC+19. Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Zero-knowledge proofs on secret-shared data via fully linear PCPs. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 67–97, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.

BCE+23. Chris Brzuska, Geoffroy Couteau, Christoph Egger, Pihla Karanko, and Pierre Meyer. New random oracle instantiations from extremely lossy functions. Cryptology ePrint Archive, Paper 2023/1145, 2023. https://eprint.iacr.org/2023/1145.

BCG+19. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 489–518, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.

BCG+20. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Correlated pseudorandom functions from variable-density LPN. In *61st FOCS*, pages 1069–1080, Durham, NC, USA, November 16–19, 2020. IEEE Computer Society Press.

BCG+22. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. Correlated pseudorandomness from expand-accumulate codes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 603–633, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Heidelberg, Germany.

BCJZ21. Jacqueline Brendel, Cas Cremers, Dennis Jackson, and Mang Zhao. The provable security of Ed25519: Theory and practice. In *2021 IEEE Symposium on Security and Privacy*, pages 1659–1676, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press.

BDD23. Mihir Bellare, Hannah Davis, and Zijing Di. Hardening signature schemes via derive-then-derandomize: Stronger security proofs for EdDSA. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part I*, volume 13940 of *LNCS*, pages 223–250, Atlanta, GA, USA, May 7–10, 2023. Springer, Heidelberg, Germany.

BDL+11. Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES 2011*, volume 6917 of *LNCS*, pages 124–142, Nara, Japan, September 28 – October 1, 2011. Springer, Heidelberg, Germany.

BHH+15. Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. SPHINCS: Practical stateless hash-based signatures. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 368–397, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

BHK+24. Fabrice Benhamouda, Shai Halevi, Hugo Krawczyk, Yiping Ma, and Tal Rabin. SPRINT: High-throughput robust distributed schnorr signatures. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024*, pages 62–91. Springer Nature Switzerland, 2024.

BLSW24. Renas Bacho, Julian Loss, Gilad Stern, and Benedikt Wagner. HARTS: High-threshold, adaptively secure, and robust threshold schnorr signatures. Cryptology ePrint Archive, Paper 2024/280, 2024. https://eprint.iacr.org/2024/280.

BMRS21. Carsten Baum, Alex J. Malozemoff, Marc B. Rosen, and Peter Scholl. Mac'n'cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 92–122, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany.

BP23. Luís T.A.N. Brandão and René Peralta. NIST first call for multi-party threshold schemes (initial public draft). Technical report, National Institute of Standards and Technology, 2023.

BST21. Charlotte Bonte, Nigel P Smart, and Titouan Tanguy. Thresholdizing HashEdDSA: MPC to the rescue. *International Journal of Information Security*, 20(6):879–894, 2021.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145, Las Vegas, NV, USA, October 14–17, 2001. IEEE Computer Society Press.

Car24. Cardano settlement layer documentation, Accessed at 2024. https://cardanodocs.com/cardano/addresses/.

CCL+19. Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Two-party ECDSA from hash proof systems and efficient instantiations. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 191–221, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.

CD23. Geoffroy Couteau and Clément Ducros. Pseudorandom correlation functions from variable-density LPN, revisited. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part II*, volume 13941 of *LNCS*, pages 221–250, Atlanta, GA, USA, May 7–10, 2023. Springer, Heidelberg, Germany.

CGG+20. Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1769–1787, Virtual Event, USA, November 9–13, 2020. ACM Press.

CGRS23. Hien Chu, Paul Gerhart, Tim Ruffing, and Dominique Schröder. Practical Schnorr threshold signatures without the algebraic group model. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 743–773, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Heidelberg, Germany.

CKM23. Elizabeth C. Crites, Chelsea Komlo, and Mary Maller. Fully adaptive Schnorr threshold signatures. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 678–709, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Heidelberg, Germany.

dec24. Decred - autonomous digital currency, Accessed at 2024. https://www.decred.org/.

Des88. Yvo Desmedt. Society and group oriented cryptography: A new concept. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 120–127, Santa Barbara, CA, USA, August 16–20, 1988. Springer, Heidelberg, Germany.

DKLs18. Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Secure two-party threshold ECDSA from ECDSA assumptions. In *2018 IEEE Symposium on Security and Privacy*, pages 980–997, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press.

DKLs19. Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Threshold ECDSA from ECDSA assumptions: The multiparty case. In *2019 IEEE Symposium on Security and Privacy*, pages 1051–1066, San Francisco, CA, USA, May 19–23, 2019. IEEE Computer Society Press.

DKLS24. Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Threshold ECDSA in three rounds. In *2024 IEEE Symposium on Security and Privacy (S&P)*. IEEE Computer Society, may 2024.

EGK+20. Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved primitives for MPC over mixed arithmetic-binary circuits. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 823–852, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany.

GG18. Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1179–1194, Toronto, ON, Canada, October 15–19, 2018. ACM Press.

GJKR96. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold DSS signatures. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 354–371, Saragossa, Spain, May 12–16, 1996. Springer, Heidelberg, Germany.

GKMN21. François Garillot, Yashvanth Kondi, Payman Mohassel, and Valeria Nikolaenko. Threshold Schnorr with stateless deterministic signing from standard assumptions. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 127–156, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany.

hyp24. Hyperledger – open source blockchain technologies, Accessed at 2024. https://www.hyperledger.org/.

JKO13. Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 955–966, Berlin, Germany, November 4–8, 2013. ACM Press.

JL17. Simon Josefsson and Ilari Liusvaara. Edwards-curve digital signature algorithm (EdDSA). In *Internet Research Task Force, Crypto Forum Research Group, RFC*, volume 8032, 2017.

KG20. Chelsea Komlo and Ian Goldberg. FROST: Flexible round-optimized Schnorr threshold signatures. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *SAC 2020*, volume 12804 of *LNCS*, pages 34–65, Halifax, NS, Canada (Virtual Event), October 21-23, 2020. Springer, Heidelberg, Germany.

KG24. Chelsea Komlo and Ian Goldberg. Arctic: Lightweight and stateless threshold schnorr signatures. Cryptology ePrint Archive, Paper 2024/466, 2024. https://eprint.iacr.org/2024/466.

KOR23. Yashvanth Kondi, Claudio Orlandi, and Lawrence Roy. Two-round stateless deterministic two-party Schnorr signatures from pseudorandom correlation functions. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 646–677, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Heidelberg, Germany.

KOS16. Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 830–842, Vienna, Austria, October 24–28, 2016. ACM Press.

Lin17. Yehuda Lindell. Fast secure two-party ECDSA signing. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 613–644, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

LN18. Yehuda Lindell and Ariel Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1837–1854, Toronto, ON, Canada, October 15–19, 2018. ACM Press.

MOR01. Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures: Extended abstract. In Michael K. Reiter and Pierangela Samarati, editors, *ACM CCS 2001*, pages 245–254, Philadelphia, PA, USA, November 5–8, 2001. ACM Press.

MPSW19. Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple schnorr multi-signatures with applications to bitcoin. *DCC*, 87(9):2139–2164, 2019.

MR01. Philip D. MacKenzie and Michael K. Reiter. Two-party generation of DSA signatures. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 137–154, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.

Nat19. National Institute of Standards and Technology (NIST). FIPS PUB 186-5 (Draft): Digital Signature Standard (DSS). https://doi.org/10.6028/NIST.FIPS.186-5, 2019.

NRSW20. Jonas Nick, Tim Ruffing, Yannick Seurin, and Pieter Wuille. MuSig-DN: Schnorr multi-signatures with verifiably deterministic nonces. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1717–1731, Virtual Event, USA, November 9–13, 2020. ACM Press.

PLD+11. Bryan Parno, Jacob R. Lorch, John R. Douceur, James W. Mickens, and Jonathan M. McCune. Memoir: Practical state continuity for protected modules. In *2011 IEEE Symposium on Security and Privacy*, pages 379–394, Berkeley, CA, USA, May 22–25, 2011. IEEE Computer Society Press.

QYYZ22. Zhi Qiu, Kang Yang, Yu Yu, and Lijing Zhou. Maliciously secure multiparty PSI with lower bandwidth and faster computation. In Cristina Alcaraz, Liqun Chen, Shujun Li, and Pierangela Samarati, editors, *ICICS 22*, volume 13407 of *LNCS*, pages 69–88, Canterbury, UK, September 5–8, 2022. Springer, Heidelberg, Germany.

RRJ+22. Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch, and Dominique Schröder. ROAST: Robust asynchronous schnorr threshold signatures. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 2551–2564, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.

SG98. Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 1–16, Espoo, Finland, May 31 – June 4, 1998. Springer, Heidelberg, Germany.

Sho00. Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 207–220, Bruges, Belgium, May 14–18, 2000. Springer, Heidelberg, Germany.

STA19. Nigel P Smart and Younes Talibi Alaoui. Distributing any elliptic curve based protocol. In *IMA International Conference on Cryptography and Coding*, pages 342–366. Springer, 2019.

Ste24. Stellar developer - security, Accessed at 2024. https://www.stellar.org/developers/guides/security.html.

XAX+21. Haiyang Xue, Man Ho Au, Xiang Xie, Tsz Hon Yuen, and Handong Cui. Efficient online-friendly two-party ECDSA signature. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 558–573, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press.

Zca24. Switch to Ed25519 for cryptographic binding of joinsplits to transactions, Accessed at 2024. https://github.com/ebfull/zcash/commit/320f2cc7e032d4047b99ab8bf7714cb8b69df8e0.

Supplementary Material

A Multi-Verifier Check Multiplication Subprotocol

Functionality $\mathcal{F}_{\mathsf{MV\text{-}CheckMULs}}$

This functionality runs with $\mathcal{P}$ and N verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_N$, and inherit all the features of PCF (shown in Fig. 7) and $\mathcal{F}_{\mathsf{MV\text{-}ND}}$ (shown in Fig. 12). Let $\mathbb{H} \subset [n]$ be the set of honest verifiers. Furthermore, this functionality is invoked by the following commands.

Check Multiplications: Upon receiving t multiplication triples $(\mathsf{CheckMuls}, sid, \{[\![\omega_{\alpha,j}]\!]_2, [\![\omega_{\beta,j}]\!]_2, [\![\omega_{\gamma,j}]\!]_2\}_{j\in[t]})$ from $\mathcal{P}$ and $\mathcal{V}$s, where t multiplication tuples are defined in multi-verifier IT-MACs. If for any $j \in [t]$ s.t. $\omega_{\gamma,j} \neq \omega_{\alpha,j} \cdot \omega_{\beta,j}$, then set $res = \mathsf{false}$. Otherwise, set $res = \mathsf{true}$. Send the set of $\{res_i\}$ to the adversary where $i \in [N] - \mathbb{H}$ and $\mathcal{V}_i$'s authenticated shares cause failure. For each $i \in \mathbb{H}$, wait for an input from the adversary and perform as follows:

- If it is $\mathsf{continue}_i$, send res to $\mathcal{V}_i$.
- If it is abort_i, send abort to $\mathcal{V}_i$.

Fig. 16: The Multi-Verifier Check Multiplications Functionality

Fig. 18 presents a secure instantiation protocol $\mathsf{CheckMuls}$ for the multi-verifier check multiplications macro in Section 4.2 and Fig. 10. We follow the paradigm of $\mathsf{AssertMultVec}$ protocol of [BMRS21] and design a multi-verifier protocol. The security of ideal functionality $\mathcal{F}_{\mathsf{MV\text{-}CheckMULs}}$ is similar to one-verifier protocol, except that we allow an adversary to control which honest verifier aborts while other verifiers output results. Fig. 17 gives an example of $\mathsf{CheckMuls}$ when $t = 2^4$.

Communication and Round Complexity. There are $\log(t)$ iterations when checking t multiplication triples. In each iteration, $\mathcal{P}$ first sends $d_{c,0}, d_{c,1}, d_{c,2} \in \mathbb{F}_{2^\kappa}$ to all $\mathcal{V}$s during the $\mathsf{Assign}(\cdot)$ macro. Then, $\mathcal{P}$ sends the MAC tag $m_i \in \mathbb{F}_{2^\kappa}$ to the verifier $\mathcal{V}_i$ during the execution of the $\mathsf{CheckZero}(\cdot)$ macro. Throughout, all verifiers only receive the data and perform computations and verifications locally. As a result, the communication cost for each iteration is $4 \cdot \kappa$ in one round. Since $\mathcal{P}$ does not need to wait for any messages from the verifiers, and the iteration structure is publicly known among all participants, $\mathcal{P}$ can merge all $\log(t)$ iterations into a single communication round. The proof is given as $\{\{(d^h_{c,0}, d^h_{c,1}, d^h_{c,2}), m^h\}_{h\in[\log(t)]}\}$, with a size of $\log(t) \cdot 4\kappa$ bits. After these iterations, the final communications involve four Assign macros, two open operations, and three $\mathsf{CheckZero}$ macros. All of these follow the same pattern, where $\mathcal{P}$ sends elements to each verifier, who performs computations locally. The complete proof

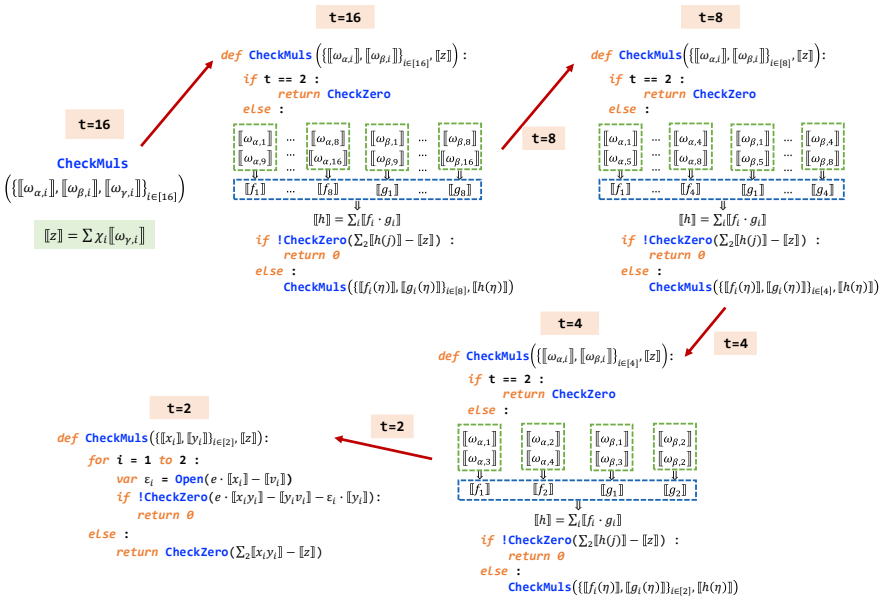

Fig. 17: An Example of CheckMuls When $t = 2^4$ Multiplication Triples Given

can then be appended to $\{\{(d^h_{c,0}, d^h_{c,1}, d^h_{c,2}), m^h_i\}_{h\in[\log(t)]}, \{d_{z,h}, d_{\hat{z},h}, \varepsilon_h, m^h_{ez,i}\}_{h\in[2]}, m_{z,i}\}$, with a total length of $\log(t)\cdot 4\kappa + 9\kappa$ bits, all sent in one round.

Lemma 1. *If the one-verifier protocol AssertMultVec passes, then the input commitments have the required relation except with probability* $\frac{t+4\log t+1}{2^\kappa-2}$.

Proof. The proof follows from [[BMRS21], Theorem.4] where the case $\sum_{i\in[t]} \omega_{\alpha,i}\cdot \omega_{\beta,i}\cdot\chi_i \neq \sum_{i\in[t]}\omega_{\gamma,i}\cdot\chi_i$ has soundness error $\frac{t+4\log t+1}{2^\kappa-2}$. □

Based on Lemma 1, we then have the following theorem.

Theorem 4. CheckMuls *UC-realizes* $\mathcal{F}_{\text{MV-CheckMULs}}$ *in the presence of an adversary statically corrupting up to* $N-1$ *verifiers, in the* PCF *assumption.*

Proof. The security of CheckMuls shown in Fig. 18 in the presence of N malicious parties crucially depends on the iterative check procedure. As in previous work [BBC+19,BMRS21], we first consider the case of a malicious prover and $N-1$ malicious verifiers, then consider an honest prover and N malicious verifiers. In each case, we always implicitly assume that $\mathcal{S}$ passes all communication between adversary $\mathcal{A}$ and environment $\mathcal{Z}$. Besides, $\mathcal{S}$ is given access to functionality $\mathcal{F}_{\text{MV-CheckMULs}}$, which runs an adversary $\mathcal{A}$ as a subroutine when emulating PCF and RO. In both cases, we show that no environment $\mathcal{Z}$ can distinguish the real-world execution from the ideal-world execution.

Protocol CheckMuls

The CheckMuls is a subroutine for $\prod_{\mathsf{MV\text{-}ND}}$. The prover $\mathcal{P}$ and N verifier $\mathcal{V}_1, \ldots, \mathcal{V}_N$ perform the consistency check on t multiplication triples each of the form $([\![\omega_{\alpha,i}]\!]_2, [\![\omega_{\beta,i}]\!]_2, [\![\omega_{\gamma,i}]\!]_2)$, for $i \in [t]$. All the parties execute as follows:

1. Each verifier $\mathcal{V}_i, i \in [N]$ and $\mathcal{P}$ inherit the string as $str := \mathsf{H}(\mathsf{mvnd}||\mathsf{sid}||k^*||\{d_j\}_{j\in[t]})$ [a], run $\mathsf{H}(str)$ to generate common randoms $\chi_1, \ldots, \chi_t \in \mathbb{F}_{2^\kappa}$, and append $str := str||\chi_1||\ldots||\chi_t$. The parties randomize $[\![z]\!]_2 = \sum_{i\in[t]} \chi_i \cdot [\![\omega_{\gamma,i}]\!]_2$ and $[\![\hat{\omega}_{\alpha,i}]\!]_2 = \chi_i \cdot [\![\omega_{\alpha,i}]\!]_2$ for $i \in [t]$.
2. While $t > 2$:
   (a) Set $t = \frac{t}{2}$. All parties define t polynomial shares as $[\![f_1]\!]_2, \ldots, [\![f_t]\!]_2 \in \mathbb{F}_{2^\kappa}[X]$ and another t polynomial shares $[\![g_1]\!]_2, \ldots, [\![g_t]\!]_2 \in \mathbb{F}_{2^\kappa}[X]$ such that $[\![f_i(j)]\!]_2 = [\![\hat{\omega}_{\alpha,j\times t+i}]\!]_2, [\![g_i(j)]\!]_2 = [\![\omega_{\beta,j\times t+i}]\!]_2$ for $j \in \{0,1\}, i \in \{1,\ldots,t\}$.
   (b) $\mathcal{P}$ generalizes a polynomial as $h = \sum_{i\in[t]} f_i \cdot g_i \in \mathbb{F}_{2^\kappa}[X]$. Note that h has a degree ≤ 2.
   (c) Let $\{c_0, c_1, c_2\}$ being the coefficients of h, $\mathcal{P}$ and $\mathcal{V}$s execute $\mathsf{Assign}(c_j)$ and define $[\![h]\!]_2$ using $[\![c_j]\!]_2, j \in \{0,1,2\}$. (Denote the transcripts sent by $\mathcal{P}$ here as $d_{c,j}$, for $j \in \{0,1,2\}$)
   (d) The parties run $\mathsf{CheckZero}(\sum_{j=1}^{2} [\![h(j)]\!]_2 - [\![z]\!]_2)$. If this check fails, the parties output false and abort.
   (e) Each verifier $\mathcal{V}_i$ and $\mathcal{P}$ append $str := str||\{d_{c,j}\}_{j\in\{0,1,2\}}$, runs $\mathsf{H}(str)$ to generate a common random $\eta \in \mathbb{F}_{2^\kappa}$ and append $str := str||\eta$.
   (f) All parties locally evaluate $[\![f_1(\eta)]\!]_2, \ldots, [\![f_t(\eta)]\!]_2$, $[\![g_1(\eta)]\!]_2, \ldots, [\![g_t(\eta)]\!]_2$ and $[\![h(\eta)]\!]_2$. They recursively back to 2.(a) until $t \leq 2$.
3. Now $\mathcal{P}$ and $\mathcal{V}$s have at most two multiplication triples, denoted as $([\![x_i]\!]_2, [\![y_i]\!]_2, [\![z]\!]_2)$, for $i \in [t]$ and $t \leq 2$. They check the validity as follows:
   (a) For $i \in [t]$, all parties generate authenticated random using $[\![v_i]\!]_2 \leftarrow \mathsf{Random}$ and compute

$$[\![z_i]\!]_2 = \mathsf{Assign}(x_i \cdot y_i), [\![\hat{z}_i]\!]_2 = \mathsf{Assign}(y_i \cdot v_i)$$

   (Denote the transcripts sent by $\mathcal{P}$ here as $\{d_{z,i}, d_{\hat{z},i}\}_{i\in[t]}$)
   (b) Each verifier $\mathcal{V}_i$ and $\mathcal{P}$ append $str := str||\{d_{z,i}||d_{\hat{z},i}\}_{i\in[t]}$, runs $\mathsf{H}(str)$ to generate common random $e \in \mathbb{F}_{2^\kappa}$ and append $str := str||e$.
   (c) For $i \in [t]$, the parties open ε_i with $[\![\varepsilon_i]\!]_2 = e \cdot [\![x_i]\!]_2 - [\![v_i]\!]_2$ and run $\mathsf{CheckZero}(e \cdot [\![z_i]\!]_2 - [\![\hat{z}_i]\!]_2 - \varepsilon_i \cdot [\![y_i]\!]_2)$.
   (d) All parties run $\mathsf{CheckZero}(\sum_{i\in[t]} [\![z_i]\!]_2 - [\![z]\!]_2)$. If any checking fails, the parties output false. Otherwise, they output true and pass the string parameter str to $\prod_{\mathsf{MV\text{-}ND}}$.

[a] The keys and transcripts, i.e., $\mathsf{sid}, k^*, \{d_j\}_{j\in[t]}$ are defined $\prod_{\mathsf{MV\text{-}ND}}$

Fig. 18: Multi-Verifier Check Multiplication Subprotocol

Malicious prover. Assume that if $\mathcal{P}$ and $N-1$ verifiers are corrupted, without loss of generality, we let $\mathcal{V}_1$ denote the honest verifier and other $\mathcal{V}_i, i \in [2, N]$ denote the corrupted verifiers. $\mathcal{S}$ interacts with $\mathcal{A}$ as follows:

1. In the input phase, $\mathcal{S}$ sampling "dummy" global key $\Delta^1 \leftarrow \mathbb{F}_{2^\kappa}$ and simulates PCF for $\mathcal{A}$ by recording all the values $\{\omega_{\alpha,i}, \omega_{\beta,i}, \omega_{\gamma,i}\}_{i\in[t]}$ and their corresponding MAC tags received from the adversary $\mathcal{A}$. Note that these values and MAC tags naturally define corresponding MAC keys. Furthermore, $\mathcal{S}$ sends $\chi_1, \ldots, \chi_t \leftarrow \mathbb{F}_{2^\kappa}$ to $\mathcal{A}$ and computes $[\![z]\!]_2, \{[\![\hat{\omega}_{\alpha,i}]\!]_2\}_{i\in[t]}$ honestly.
2. While $t > 2$:
   (a) $\mathcal{S}$ executes Step 4.(a)-(f) honestly as an honest verifier, except that $\mathcal{S}$ checks the zero-sharing using transcripts of corrupted $\mathcal{P}$, i.e., checking whether what it received from $\mathcal{A}$ equal to $\sum_{j=1}^{2} k_h(j) - k_z$ using "dummy" global key and local key k_h of the polynomial h. This equation is checkable as $\mathcal{S}$ knows all the secret values $\{\omega_{\alpha,i}, \omega_{\beta,i}, \omega_{\gamma,i}\}_{i\in[\tau]}$ and $\mu_j, j \in \{0, 1, 2\}$ in the Assign subroutine.
3. For $i \in [t]$, $\mathcal{S}$ simulates RO by receiving $v_i, m_{v,i}$ from $\mathcal{A}$ and computes their corresponding "dummy" local keys.
4. $\mathcal{S}$ simulates RO for $\mathcal{A}$ by sampling uniform $e \leftarrow \mathbb{F}_{2^\kappa}$ and sending it to $\mathcal{P}$
5. $\mathcal{S}$ plays the role of the honest verifier $\mathcal{V}_1$ to perform the $\mathsf{CheckZero}$ procedures with $\mathcal{A}$, using the "dummy" global key and local keys.
6. If the honest $\mathcal{V}_1$ (simulated by $\mathcal{S}$) aborts in any $\mathsf{CheckZero}$ procedure, then $\mathcal{S}$ sends abort to $\mathcal{F}_{\mathsf{MV\text{-}CheckMULs}}$ and aborts. Otherwise, $\mathcal{S}$ sends the multiplication triples $\{[\![\omega_{\alpha,i}]\!]_2, [\![\omega_{\beta,i}]\!]_2, [\![\omega_{\gamma,i}]\!]_2\}_{i\in[\tau]}$ to functionality $\mathcal{F}_{\mathsf{MV\text{-}CheckMULs}}$ on behalf of corrupted prover $\mathcal{P}$ and corrupted verifiers $\mathcal{V}_2, \ldots, \mathcal{V}_N$.

The simulated view of $\mathcal{A}$ has the identical distribution as its view in the real execution. Note that the "dummy" global key sampled by $\mathcal{S}$ has the same distribution as the real global key, $\mathcal{S}$ emulates PCF, and, in each extend execution, the MAC tags sent to $\mathcal{A}$ are computed as follows. $\mathcal{S}$ has access to the $\mathcal{S}_\sigma$ (c.f. Definition 2) using its chosen $\mathsf{msk}^i := \Delta^i, i \in [N]$. At the beginning of each concurrent execution: $\mathcal{S}$ first compute $y_\sigma^{(j)} := \mathsf{PCF.Eval}(\sigma, k_\sigma, \mathsf{sid})$. Then it queries $\mathsf{RSample}$ with $(1^\kappa, \mathsf{msk}, \sigma, y_\sigma^{(j)})$ and receives $y_{1-\sigma}^{(j)}$ as the local MAC keys. Based on the security definition of PCF, the simulated view is computationally indistinguishable from that in the real execution. Furthermore, whenever honest verifier $\mathcal{V}_1$ in the real execution aborts, $\mathcal{S}$ acts as $\mathcal{V}_1$ in the ideal execution aborts. Thus, it remains to bound the probability that the $\mathcal{V}_1$ in the real execution accepts but the transcripts received by $\mathcal{S}$ pass the $\mathsf{CheckZero}$ subcheck. In this case, the malicious $\mathcal{P}$ will successfully trick honest $\mathcal{V}_1$ into accepting a forged MAC tag. According to Lemma 1, the probability that the honest $\mathcal{V}_1$ in the real execution doesn't abort is at most $\frac{t+4\log t+1}{2^\kappa-2}$. Thus, the output distribution of the honest verifier in the real-world execution is indistinguishable from that in the ideal-world execution.

Malicious verifiers. Assume that if $\mathcal{P}$ is honest and N verifiers are corrupted. $\mathcal{S}$ interacts with $\mathcal{A}$ as follows:

1. In the input phase, $\mathcal{S}$ emulates PCF by recording global key $\Delta^1, \ldots, \Delta^N \in \mathbb{F}_{2^\kappa}$ and the local MAC keys for all input values, which are sent by $\mathcal{A}$.
2. Upon receiving $\{[\![\omega_{\alpha,i}]\!]_2, [\![\omega_{\beta,i}]\!]_2, [\![\omega_{\gamma,i}]\!]_2\}_{i\in[t]}$, $\mathcal{S}$ sends them to $\mathcal{F}_{\mathsf{MV\text{-}CheckMULs}}$ and receives $\{res_i\}$, where $i \in \mathbb{B} \subseteq [N]$ denotes $\mathcal{V}_i$'s authenticated share cause the failure result.
3. $\mathcal{S}$ emulates RO by sending $\chi_1, \ldots, \chi_t \leftarrow \mathbb{F}_{2^\kappa}$ to $\mathcal{A}$ and computes $[\![z]\!]_2$ and $\{[\![\hat{\omega}_{\alpha,i}]\!]_2, [\![\omega_{\beta,i}]\!]_2\}_{i\in[t]}$.
4. While $t > 2$:
   (a) $\mathcal{S}$ emulates PCF by recording the local MAC keys for the random value used in the Assign subroutine, sent by $\mathcal{A}$.
   (b) $\mathcal{S}$ samples $\{d_{0,t}, d_{1,t}, d_{2,t}\}$ and sends them to $\mathcal{A}$ in the Assign subroutine. Then, it computes their MAC keys using the keys received from $\mathcal{A}$.
   (c) $\mathcal{S}$ runs the $\mathsf{CheckZero}$ subroutine with $\mathcal{A}$ according to the set of $\{res_i\}, i \in \mathbb{B}$: If res_i, $\mathcal{S}$ sends random $m^* \leftarrow \mathbb{F}_{2^\kappa}$ to $\mathcal{V}_i$; Otherwise, $\mathcal{S}$ computes and sends $\sum_{i=1}^{2} k_h(j) - k_z$ to $\mathcal{V}_i$ where the local MAC keys k_h and k_z are computed with the global keys and local keys recorded by $\mathcal{S}$.
5. For $i \in [t]$, $\mathcal{S}$ simulates PCF by receiving $k_{v,i}$ from $\mathcal{A}$.
6. $\mathcal{S}$ simulates RO for $\mathcal{A}$ by sampling uniform $e \leftarrow \mathbb{F}_{2^\kappa}$ and sending it to $\mathcal{V}$s.
7. $\mathcal{S}$ plays the role of the honest prover $\mathcal{P}$ to perform the remaining $\mathsf{CheckZero}$ procedures with $\mathcal{A}$, using the global keys and local keys received from $\mathcal{A}$. Specifically, if $res_j, j \in \mathbb{B}$, $\mathcal{S}$ sends random $m^* \leftarrow \mathbb{F}_{2^\kappa}$ to $\mathcal{V}_j$; Otherwise, $\mathcal{S}$ computes 0-sharing using $k_{v,i}, k_{x,i}, k_{y,i}, k_z$ and global keys recorded by $\mathcal{S}$, and then sends it to $\mathcal{V}_j$.

We use a hybrid argument to prove that the two worlds are computationally indistinguishable.

- **Hybrid$_0$**. This is the real world.
- **Hybrid$_1$**. This hybrid is identical to the previous one, except that $\mathcal{S}$ emulates PCF and, in each Extend execution, the transcripts $\{d_{i,0}, d_{i,1}, d_{i,2}\}_{i\in[t]}$ sent to $\mathcal{A}$ is computed as follows.
  $\mathcal{S}$ has access to the $\mathcal{S}_\sigma$ (c.f. Definition 2) using the global keys $\mathsf{msk}^i := \Delta^i, i \in [N]$ received from $\mathcal{A}$. At the beginning of each concurrent execution:
  - In the input phase, $\mathcal{S}$ first compute $\boldsymbol{k}^i := \mathsf{PCF.Eval}(\sigma, k_\sigma, \mathsf{sid})$. Then it queries $\mathsf{RSample}$ with $(1^\kappa, \mathsf{msk}^i, \sigma, \boldsymbol{k}^i)$ and receives $(\boldsymbol{x}^i, \boldsymbol{m}^i)$ where $\boldsymbol{x}^i$ denotes all the input values corresponding with $\mathcal{V}_i$ and $\boldsymbol{m}^i = \boldsymbol{k}^i + \Delta^i \cdot \boldsymbol{x}^i$.
  - For $\mathsf{Assign}(c_j)$ for $j \in \{0, 1, 2\}$, $\mathcal{S}$ computes $k^i_{\mu,j} := \mathsf{PCF.Eval}(\sigma, k_\sigma, \mathsf{sid} + \mathsf{j})$. Then it queries $\mathsf{RSample}$ with $(1^\kappa, \mathsf{msk}^i, \sigma, k^i_{\mu,j})$ and receives $(\mu^i_j, m^i_{\mu,j})$ where μ^i_j denotes the authenticated random values used in Assign subroutine and $m^i_{\mu,j} = k^i_{\mu,j} + \Delta^i \cdot \mu^i_j$.
  - For the final t multiplication triples, i.e., $\mathsf{Assign}(x_j \cdot y_j)$ and $\mathsf{Assign}(y_j \cdot v_j)$ for $j \in [t]$, $\mathcal{S}$ similarly compute

$$k^i_{xy,j} := \mathsf{PCF.Eval}(\sigma, k_\sigma, \mathsf{sid} + \log \mathsf{t} + 2\mathsf{j}),$$
$$k^i_{yv,j} := \mathsf{PCF.Eval}(\sigma, k_\sigma, \mathsf{sid} + \log \mathsf{t} + 2\mathsf{j} + 1).$$

Then it queries RSample with the following operations:

$$(\mu^i_{xy,j}, m^i_{xy,j}) := \mathsf{RSample}((1^\kappa, \mathsf{msk}^i, \sigma, k^i_{xy,j}),$$
$$(\mu^i_{yv,j}, m^i_{yv,j}) := \mathsf{RSample}((1^\kappa, \mathsf{msk}^i, \sigma, k^i_{yv,j}).$$

- In the rest of the execution, $\mathcal{S}$ uses these oracle responses to transcript the honest $\mathcal{P}$.

The resulting view is equivalently defined as in the previous hybrid under the security definition of PCF (c.f. Definition 2) with probability $\mathsf{negl}(\kappa)$. Therefore, this hybrid is computationally indistinguishable from the previous one.

– **Hybrid$_2$**. This hybrid is identical to the previous one, except that PCF accesses the $\mathcal{Y}$-function to generate the authenticated secret shares instead of the real-world PCF function. It follows from the assumption of pseudorandom $\mathcal{Y}$-correlated outputs that this hybrid is computationally indistinguishable from the previous one.

– **Hybrid$_3$**. This hybrid is identical to the previous one, except that in each iteration execution, i.e., while $t > 2$, $\mathcal{S}$ replaces $\{d_{0,t}, d_{1,t}, d_{2,t}\}$ and $\{d_{xy,i}, d_{yv,i}\}$ by random values. Observe that in each Assign subroutine, the difference d-values are pseudorandomly due to the $\mathcal{Y}$-function and serve as pseudorandom one-time pad for all the coefficients $\{c_{0,t}, c_{1,t}, c_{2,t}\}$ and products $\{x_i \cdot y_i, y_i \cdot v_i\}$. Therefore, this hybrid is computationally indistinguishable from the previous one.

– **Hybrid$_4$**. This hybrid is identical to the previous one, except that $\mathcal{S}$ emulates PCF and, upon receiving t multiplication triples $\{[\![\omega_{\alpha,i}]\!]_2, [\![\omega_{\beta,i}]\!]_2, [\![\omega_{\gamma,i}]\!]_2\}_{i\in[t]}$, it follows protocol CheckMuls to run CheckZero and control the output of the corrupted $\mathcal{V}_i, i \in \mathbb{B} \subseteq [N]$ by using the transcripts received from $\mathcal{A}$, and the responses from $\mathcal{F}_{\mathsf{MV\text{-}CheckMULs}}$. This hybrid is computationally indistinguishable from the previous one: (1) if $\mathcal{V}_i, i \in \mathbb{B} \subseteq [N]$, the polynomial points evaluated by random $\eta \in \mathbb{F}_{2^\kappa}$ essentially serve as the one-time pad for MAC tags of $\sum_{j=1}^{2} h(j) - z$, except with collision probability of $\frac{1}{q}$. (2) if $\mathcal{V}_i, i \notin \mathbb{B}$, the output of $\mathcal{V}_i$ is exactly as expected.
It is clear that this hybrid is the ideal world.

The above hybrid argument completes the proof. □

B Security Proof of Theorem 1

This section presents the security proof of the generation protocol of mv-edaBits, as shown in Fig. 11. The high-level of this security proof includes *correctness* and *privacy*, i.e. firstly, a correct mv-edaBits should be generated after the protocol $\prod_{\mathsf{MV\text{-}edaBits}}$; secondly, $\mathcal{A}$ will extract no additional information. The *privacy* will be proven via UC-framework 2.2 when $\mathcal{A}$ has corrupted up to $n-1$ parties. The core challenge is to prove that if the random ρs are inconsistent between the boolean and arithmetic fields, one can find it with non-negligible probability.

Theorem 5. *(Restate of Theorem 1) Protocol* $\prod_{\mathsf{MV\text{-}edaBits}}$ *UC-realizes* $\mathcal{F}_{\mathsf{MV\text{-}edaBits}}$ *in the presence of an adversary statically corrupting up to* $n-1$ *parties, in the* $\mathcal{F}_{\mathsf{MV\text{-}CheckMULs}}$*-hybrid random oracle model and* PCF *assumption.*

Proof. First, we analyze its correctness as follows: The correctness of $\prod_{\mathsf{MV\text{-}edaBits}}$ protocol as described in Fig. 11 relies on the gate-by-gate evaluation and consistency check. In the honest case, all the parties obtain $\boldsymbol{b} := \{\boldsymbol{b}[1], \ldots, \boldsymbol{b}[\ell]\}$ such that $\sum_{j=1}^{\ell} 2^{j-1} \cdot \boldsymbol{b}[j] = \sum_{j=1}^{\ell} 2^{j-1} \cdot \boldsymbol{a}[j] + \chi \cdot \sum_{j=1}^{\ell} 2^{j-1} \cdot \boldsymbol{\rho}[j] \mod q$ and therefore

$$b = (\sum_{j=1}^{\ell} 2^{j-1} \cdot \boldsymbol{b}[j] \mod q) = (r + \chi \cdot \rho) \mod q$$
$$0 = r + \chi \cdot \rho - b$$

with r and ρ are boolean parts of mv-edaBits, r and ρ are integer parts of mv-edaBits, for any χ unpredictably generated by a hash chain as $\mathsf{H}(str||\mathsf{H}(str||1)||\mathsf{H}(str||2))$.

In the following, we prove the security of our $\prod_{\mathsf{MV\text{-}edaBits}}$ protocol in the multi-party malicious setting. We consider the case that $n-1$ parties are corrupted. We always implicitly assume that $\mathcal{S}$ passes all communication between adversary $\mathcal{A}$ and environment $\mathcal{Z}$. Specifically, $\mathcal{S}$ separately simulates honest $\mathcal{P}$ in the case of malicious verifiers and honest one verifier $\mathcal{V}^*$ in the case of malicious prover. Our goal is to prove that $\mathcal{S}$, knowing publicly available information and the adversary's input, could produce a view of $\mathcal{A}$ (including transmitted message and output) that is indistinguishable from that produced within the real world, then proves that $\mathcal{A}$ does not know additional information in the secure protocol. Thus, $\mathcal{S}$ must extract the corrupted prover's witness or corrupted verifier's global key to send to the trusted party. This is possible because in the $(\mathcal{F}_{\mathsf{MV\text{-}CheckMULs}})$-hybrid model and PCF assumption $\mathcal{S}$ receives the secret inputs from $\mathcal{A}$.

Malicious Verifiers. Assume that if $\mathcal{P}$ is honest and N verifiers are corrupted. $\mathcal{S}$ interacts with $\mathcal{A}$ as follows:

1. In the Setup phase, $\mathcal{S}$ invokes $\exp_1^{sec}(\kappa)$ (In Fig. 5) using the global keys $\Delta^1, \ldots, \Delta^N \in \mathbb{F}_{2^\kappa}$ and $\Lambda^1, \ldots, \Lambda^N \in \mathbb{Z}_q$ sampled uniformly, receiving k_1^i for $i \in [N]$ from $\exp_1^{sec}(\kappa)$. $\mathcal{S}$ generates k^* honestly.
2. In the Create mv-edaBits phase, for given common string str:
   (a) $\mathcal{S}$ records all the randomness used by $\mathcal{A}$ from the set of queries made by $\mathcal{V}_i$ to RO.
   (b) $\mathcal{S}$ query $\exp_1^{sec}(\kappa)$ with input $\mathsf{H}(str||1)$ and $\mathsf{H}(str||2)$ to learn $[\![\rho]\!]_q$, $[\![a]\!]_q$, $[\![\boldsymbol{\rho}]\!]_2$ and $[\![\boldsymbol{a}]\!]_2$.
   (c) $\mathcal{S}$ generates χ and evaluate the circuit $\mathcal{C}_{\mathsf{aff}}$ cooperated with $\mathcal{V}$s:
      i. In the subroutine Assign, $\mathcal{S}$ receives the MAC keys for all random values (i.e., $\boldsymbol{k}_{\mu_i} \in \mathbb{F}_{2^\kappa}^{t_2}$) from $\mathcal{A}$ by emulating PCF.
      ii. $\mathcal{S}$ executes Step 4.(a) as an honest prover, except that for j-th multiplication gates, $\mathcal{S}$ samples random $d_j \leftarrow \mathbb{F}_2$ for all $j \in [t_2]$ and sends them to $\mathcal{V}$s.

iii. $\mathcal{S}$ receives $\{[\![\omega^*_{\alpha,j}]\!]_2, [\![\omega^*_{\beta,j}]\!]_2, [\![\omega^*_{\gamma,j}]\!]_2\}_{j\in[t_2]}$ from $\mathcal{A}$ on behalf of the functionality $\mathcal{F}_{\mathsf{MV\text{-}CheckMULs}}$, if $\exists j$, s.t. $[\![\omega^*_{l,j}]\!]_2 \neq [\![\omega_{l,j}]\!]_2, l \in \{\alpha, \beta, \gamma\}$ where $[\![\omega_{l,j}]\!]_2$ is computed by $\mathcal{S}$ following the protocol, sends false on behalf of $\mathcal{F}_{\mathsf{MV\text{-}CheckMULs}}$ and aborts.

(d) $\mathcal{S}$ computes $b = a + \chi \cdot \rho$ and reveals its binary representation $\boldsymbol{b}$ (along with their MAC tags as $\boldsymbol{m}[i]^j = \boldsymbol{k}[i]^j + \Delta^j \cdot \boldsymbol{b}[i]$, for $j \in [N]$) to all the verifiers.

(e) $\mathcal{S}$ computes z_i, the secret shares of $[\![a]\!]_q + \chi \cdot [\![\rho]\!]_q - b$ for each $\mathcal{V}_i$ (this is computable as it knows Λ^i, ρ, a and their corresponding MAC tags). $\mathcal{S}$ sends z_i to the corresponding $\mathcal{V}_i, i \in [N]$ on behalf of $\mathcal{P}$.

(f) $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

We use a hybrid argument to prove that the two worlds are computationally indistinguishable.

– **Hybrid$_0$**. This is the real world.

– **Hybrid$_1$**. This hybrid is identical to the previous one, except that $\mathcal{S}$ emulates PCF . Specifically, in each edaBits creation execution, $\mathcal{S}$ has access to the $\mathcal{S}_\sigma$ (c.f. Definition 2) using the global keys $\mathsf{msk}^i := \Delta^i, i \in [N]$ received from $\mathcal{A}$. The different executions are:

- $\mathcal{S}$ firstly computes the MAC keys as $\boldsymbol{k}^i_\rho := \mathsf{PCF.Eval}(\sigma, k_\sigma, \mathsf{H}(str||1)$ and $\boldsymbol{k}^i_a := \mathsf{PCF.Eval}(\sigma, k_\sigma, \mathsf{H}(str||2))$. Then it queries $\mathsf{RSample}$ (in Fig. 6) with $(1^\kappa, \mathsf{msk}^i, \sigma, \boldsymbol{k}^i_\rho)$ and $(1^\kappa, \mathsf{msk}^i, \sigma, \boldsymbol{k}^i_a)$. $\mathcal{S}$ will receives $(\boldsymbol{\rho}, \boldsymbol{m}^i_\rho)$ and $(\boldsymbol{a}, \boldsymbol{m}^i_a)$.
- For the t_2 multiplication triples, i.e., $\mathsf{Assign}(x_j \cdot y_j)$ and $\mathsf{Assign}(y_j \cdot v_j)$ for $j \in [t_2]$, $\mathcal{S}$ similarly compute

$$k^i_{xy,j} := \mathsf{PCF.Eval}(\sigma, k_\sigma, \mathsf{H}(str||2j)),$$
$$k^i_{yv,j} := \mathsf{PCF.Eval}(\sigma, k_\sigma, \mathsf{H}(str||2j+1)).$$

  Then it queries $\mathsf{RSample}$ with the following operations:

$$(\mu^i_{xy,j}, m^i_{xy,j}) := \mathsf{RSample}((1^\kappa, \mathsf{msk}^i, \sigma, k^i_{xy,j}),$$
$$(\mu^i_{yv,j}, m^i_{yv,j}) := \mathsf{RSample}((1^\kappa, \mathsf{msk}^i, \sigma, k^i_{yv,j}).$$

- In the rest of the execution, $\mathcal{S}$ uses these oracle responses to transcript the honest $\mathcal{P}$.

The resulting view is equivalently defined as in the previous hybrid under the security definition of PCF (c.f. Definition 2) with probability $\mathsf{negl}(\kappa)$. Therefore, this hybrid is computationally indistinguishable from the previous one.

– **Hybrid$_2$**. This hybrid is identical to the previous one, except that PCF accesses the $\mathcal{Y}$-function to generate the authenticated secret shares instead of the real-world PCF function. It follows from the assumption of pseudorandom $\mathcal{Y}$-correlated outputs that this hybrid is computationally indistinguishable from the previous one.
It is clear that this hybrid is the ideal world.

The above hybrid argument completes the proof.

Malicious Prover. We consider the case that $n-1$ parties are corrupted. Without loss of generality, we let $\mathcal{V}_1$ denote the honest verifier and $\mathcal{V}_i, i \in \mathcal{I} = [2,n]$ are corrupted. Also, $\mathcal{P}$ is malicious in this simulation. $\mathcal{S}$ interacts with $\mathcal{A}$ as follows:

1. In the Setup phase, $\mathcal{S}$ receives Δ^1 from mv-edaBits and samples other global keys $\Delta^2, \ldots, \Delta^N \in \mathbb{F}_{2^\kappa}$. $\mathcal{S}$ sends them to invokes $\exp_1^{sec}(\kappa)$ (In Fig. 5) using the global keys and receives k_0^i for $i \in [N]$ and k_1^i for $i \in [2,N]$ from $\exp_1^{sec}(\kappa)$. $\mathcal{S}$ generates k^* honestly.
2. In the Create mv-edaBits phase, for given common string str:
   (a) $\mathcal{S}$ records all the randomness used by $\mathcal{A}$ from the set of queries made by $\mathcal{P}$ and $\mathcal{V}_i, i \in [2,N]$ to RO.
   (b) $\mathcal{S}$ records all the edaBits values (i.e., $\boldsymbol{\rho}, \boldsymbol{a} \in \mathbb{F}_2^\ell, \rho, a \in \mathbb{Z}_q$) and corresponding MAC tags (i.e., $\boldsymbol{m}_\rho^j, \boldsymbol{m}_a^j \in \mathbb{F}_{2^\kappa}^\ell, m_\rho^j, m_a^j \in \mathbb{Z}_q$ for $j \in [1,N]$), which are extracted by emulating PCF for corrupted $\mathcal{P}$. $\mathcal{S}$ defines the corresponding MAC keys of edaBits, i.e., $\boldsymbol{k}_\rho^j, \boldsymbol{k}_a^j \in \mathbb{F}_{2^\kappa}^\ell, k_\rho^j, k_a^j \in \mathbb{Z}_q$).
   (c) $\mathcal{S}$ evaluate the circuit $\mathcal{C}_{\mathsf{aff}}$ cooperated with other parties:
      – In the subroutine Assign, $\mathcal{S}$ records all the values (i.e., $\boldsymbol{\mu}[1], \ldots, \boldsymbol{\mu}[t_2] \in \mathbb{F}_2$) and their corresponding MAC tags (i.e., $\boldsymbol{m}_\mu \in \mathbb{F}_{2^\kappa}^{t_2}$) by emulating PCF for $\mathcal{A}$. Here, $\mathcal{S}$ can define the corresponding MAC keys of these values (i.e., $\boldsymbol{k}_\mu \in \mathbb{F}_{2^\kappa}^{t_2}$).
      – $\mathcal{S}$ executes Step 4.(a) as an honest verifier.
      – $\mathcal{S}$ receives $\{[\![\omega_{\alpha,j}^*]\!]_2, [\![\omega_{\beta,j}^*]\!]_2, [\![\omega_{\gamma,j}^*]\!]_2\}_{j\in[t_2]}$ from $\mathcal{A}$ on behalf of the functionality $\mathcal{F}_{\text{MV-CheckMULs}}$, if $\forall j$, s.t. $\omega_{\alpha,j}^* \cdot \omega_{\beta,j}^* = \omega_{\gamma,j}^*$, and $\exists i$, s.t. $[\![\omega_{l,i}^*]\!]_2, \in \{\alpha,\beta,\gamma\}$ is not a valid IT-MAC, sends abort to $\mathcal{V}_i$ and sends true to all the other $\mathcal{V}_j, j \in [2,N]\backslash\{i\}$ on behalf of $\mathcal{F}_{\text{MV-CheckMULs}}$.
   (d) Upon opening $[\![\boldsymbol{b}]\!]_2$, if any $\boldsymbol{b}[j]$' IT-MACs is invalid, $\mathcal{S}$ aborts. Otherwise, $\mathcal{S}$ continues.
   (e) $\mathcal{S}$ aborts if $[\![0]\!]_q \neq [\![a]\!]_q + \chi \cdot [\![\rho]\!]_q - b$ revealed from $\mathcal{P}$, it is computable as $\mathcal{S}$ knows all the a, ρ, m_a, m_ρ and b.
   (f) $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

Indistinguishability of the simulation is argued as follows: the only non-syntactic difference between the simulation and the real protocol is that when $a \neq \sum_{i=1}^{\ell} 2^{i-1} \cdot \boldsymbol{a}[i]$ and $\rho \neq \sum_{i=1}^{\ell} 2^{i-1} \cdot \boldsymbol{\rho}[i]$. As the CheckMuls subroutine guarantees that the $b = a + \chi \cdot \rho$ is correctly computed from binary parts of edaBits except with the negligible probability $\mathsf{negl}(\kappa)$. Now we consider when red ρ, a (aggregated by binary representations of mv-edaBits) is not consistent with blue ρ, a (generated from integer part of mv-edaBits). Assume that $\rho = \rho + e_\rho$ and $a = a + e_a$, if CheckZero($[\![a]\!]_q + \chi \cdot [\![\rho]\!]_q - b$) successes, we have

$$\begin{aligned} 0 =& a + \chi \cdot \rho - (a + \chi \cdot \rho) \\ =& a + e_a + \chi \cdot (\rho + e_\rho) - (a + \chi \cdot \rho) \\ =& e_a + \chi \cdot e_\rho \end{aligned}$$

Since that $\mathsf{H}(str||\mathsf{H}(str||1)||\mathsf{H}(str||2)) = \chi \in \mathbb{Z}_q$ is unpredictable uniformly from RO, this equation holds with probability at most $\frac{1}{q}$. In conclusion, $\mathcal{Z}$ cannot distinguish between the real execution and ideal execution, except with probability $\mathsf{negl}(\kappa) + \frac{1}{q}$.

□

C Security Proof of Theorem 2

This section presents the security proof of multi-verifier stateless deterministic nonce derivation based on MVZK, as shown in Fig. 13. Similarly, this security proof begins by *correctness* where the nonce R could be verifiably generated. Then, the *privacy* will be proven via UC-framework 2.2 with the case that $n-1$ parties are corrupted.

Theorem 6. *(Restate of Theorem 2) Protocol* $\prod_{\mathsf{MV\text{-}ND}}$ *UC-realizes* $\mathcal{F}_{\mathsf{MV\text{-}ND}}$ *in the presence of an adversary statically corrupting up to* $n-1$ *parties, in the* $(\mathcal{F}_{\mathsf{MV\text{-}CheckMULs}}, \mathcal{F}_{\mathsf{MV\text{-}edaBits}})$*-hybrid random oracle model and* PCF *assumption.*

Proof. First, we analyze its correctness as follows.

Correctness. The correctness of MV-ND protocol $\prod_{\mathsf{MV\text{-}ND}}$ as described in Fig. 13 relies on the gate-by-gate evaluation and conversion. In the honest case, the parties obtain $\boldsymbol{c} := \{\boldsymbol{c}[1], \ldots, \boldsymbol{c}[\ell]\}$ such that $c = \sum_{j=1}^{\ell} 2^{j-1} \cdot \boldsymbol{c}[j] = \sum_{j=1}^{\ell} 2^{j-1} \cdot \boldsymbol{r}[j] + \sum_{j=1}^{\ell} 2^{j-1} \cdot \boldsymbol{\rho}[j] = r + \rho \mod q$ and therefore $[\![R]\!]_q = (c - [\![\rho]\!]_q) \cdot G = (\sum_{j=1}^{\ell} 2^{j-1} \cdot \boldsymbol{r}[j]) \cdot G$.

In the following, we prove the security of our $\prod_{\mathsf{MV\text{-}ND}}$ protocol in the multi-party malicious setting. We consider the case that $n-1$ parties are corrupted. Similar to $\prod_{\mathsf{MV\text{-}edaBits}}$, we construct the $\mathcal{S}$ that separately simulates honest $\mathcal{P}$ in the case of malicious verifiers and honest one verifier $\mathcal{V}^*$ in the case of malicious prover. Our security proof is based on the $(\mathcal{F}_{\mathsf{MV\text{-}CheckMULs}}, \mathcal{F}_{\mathsf{MV\text{-}edaBits}})$-hybrid model and PCF assumption.

Malicious Verifiers. Assume that if $\mathcal{P}$ is honest and N verifiers are corrupted. $\mathcal{S}$ interacts with $\mathcal{A}$ as follows:

1. In the Setup phase, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{MV\text{-}edaBits}}$ by invoking $\exp_1^{sec}(\kappa)$ (In Fig. 5) using the global keys $\Delta^1, \ldots, \Delta^N \in \mathbb{F}_{2^\kappa}$ and $\Lambda^1, \ldots, \Lambda^N \in \mathbb{Z}_q$ sampled uniformly, receiving k_1^i for $i \in [N]$ from $\exp_1^{sec}(\kappa)$. Note that $\mathcal{S}$ knows $[\![\mathsf{dk}]\!]_2$ acting as $\mathcal{V}$s.
2. In the Proof, for given message msg:
   (a) $\mathcal{S}$ computes $str = \mathsf{RO}(\mathsf{mvnd}||\mathsf{msg}||k^*)$. Furthermore, $\mathcal{S}$ records all the randomness used by $\mathcal{A}$ from the set of queries made by $\mathcal{V}_i, i \in [N]$ to RO.
   (b) $\mathcal{S}$ samples $\boldsymbol{\rho} \in \{0,1\}^\ell$, generates $[\![\rho]\!]_q, [\![\boldsymbol{\rho}]\!]_2$ where $\rho = \sum_{i=1}^{\ell} 2^{i-1} \cdot \boldsymbol{\rho}[i]$. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{MV\text{-}edaBits}}$ by sending $[\![\rho]\!]_q, [\![\boldsymbol{\rho}]\!]_2$ to the correspond $\mathcal{V}$s.
   (c) $\mathcal{S}$ records all the randomness used by $\mathcal{A}$ from the set of queries made by $\mathcal{V}_i$ to RO.

(d) $\mathcal{S}$ generates χ and evaluate the circuit $\tilde{\mathcal{C}}$ cooperated with $\mathcal{V}$s:
   i. In the subroutine Assign, $\mathcal{S}$ receives the MAC keys for all random values (i.e., $\boldsymbol{k}_{\mu_i} \in \mathbb{F}_{2^\kappa}^{\tilde{t}}$) from $\mathcal{A}$ by emulating PCF.
   ii. $\mathcal{S}$ executes Step 4.(a) as an honest prover, except that for j-th multiplication gates, $\mathcal{S}$ samples random $d_j \leftarrow \mathbb{F}_2$ for all $j \in [\tilde{t}]$ and sends them to $\mathcal{V}$s.
   iii. $\mathcal{S}$ receives $\{[\![\omega^*_{\alpha,j}]\!]_2, [\![\omega^*_{\beta,j}]\!]_2, [\![\omega^*_{\gamma,j}]\!]_2\}_{j\in[\tilde{t}]}$ from $\mathcal{A}$ on behalf of the functionality $\mathcal{F}_{\mathsf{MV\text{-}CheckMULs}}$, if $\exists j$, s.t. $[\![\omega^*_{l,j}]\!]_2 \neq [\![\omega_{l,j}]\!]_2, l \in \{\alpha, \beta, \gamma\}$ where $[\![\omega_{l,j}]\!]_2$ is computed by $\mathcal{S}$ following the protocol, sends false on behalf of $\mathcal{F}_{\mathsf{MV\text{-}CheckMULs}}$ and aborts.

(e) $\mathcal{S}$ samples random c and reveals its binary representation $\boldsymbol{c}$ (along with their MAC tags as $\boldsymbol{c}[i]^j = \boldsymbol{k}[i]^j + \Delta^j \cdot \boldsymbol{c}[i]$, for $j \in [N]$) to all the verifiers.

(f) $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

We use a hybrid argument to prove that the two worlds are computationally indistinguishable.

– **Hybrid$_0$**. This is the real world.

– **Hybrid$_1$**. This hybrid is identical to the previous one, except that $\mathcal{S}$ emulates PCF for the circuit evaluation. Specifically, for the $\tilde{t}$ multiplication triples, i.e., $\mathsf{Assign}(x_j \cdot y_j)$ and $\mathsf{Assign}(y_j \cdot v_j)$ for $j \in [\tilde{t}]$, $\mathcal{S}$ computes

$$k^i_{xy,j} := \mathsf{PCF.Eval}(\sigma, k_\sigma, \mathsf{H}(str||2j)),$$
$$k^i_{yv,j} := \mathsf{PCF.Eval}(\sigma, k_\sigma, \mathsf{H}(str||2j+1)).$$

Then it queries $\mathsf{RSample}$ with the following operations:

$$(\mu^i_{xy,j}, m^i_{xy,j}) := \mathsf{RSample}((1^\kappa, \mathsf{msk}^i, \sigma, k^i_{xy,j}),$$
$$(\mu^i_{yv,j}, m^i_{yv,j}) := \mathsf{RSample}((1^\kappa, \mathsf{msk}^i, \sigma, k^i_{yv,j}).$$

In the rest of the execution, $\mathcal{S}$ uses these oracle responses to transcript the honest $\mathcal{P}$.
The resulting view is equivalently defined as in the previous hybrid under the security definition of PCF (c.f. Definition 2) with probability $\mathsf{negl}(\kappa)$. Therefore, this hybrid is computationally indistinguishable from the previous one.

– **Hybrid$_2$**. This hybrid is identical to the previous one, except that PCF accesses the $\mathcal{Y}$-function to generate the authenticated secret shares instead of the real-world PCF function. It follows from the assumption of pseudorandom $\mathcal{Y}$-correlated outputs that this hybrid is computationally indistinguishable from the previous one.
It is clear that this hybrid is the ideal world.

The above hybrid argument completes the proof.

Malicious Prover. We consider the case that $n-1$ parties are corrupted. Without loss of generality, we let $\mathcal{V}_1$ denote the honest verifier and $\mathcal{V}_i, i \in \mathcal{I} = [2, n]$ are corrupted. Also, $\mathcal{P}$ is malicious in this simulation. $\mathcal{S}$ interacts with $\mathcal{A}$ as follows:

1. In the Setup phase, $\mathcal{S}$ samples $\Delta^1, \Delta^2, \ldots, \Delta^N \in \mathbb{F}_{2^\kappa}$ emulating as $\mathcal{F}_{\mathsf{MV\text{-}edaBits}}$. $\mathcal{S}$ sends them to invokes $\exp_1^{sec}(\kappa)$ (In Fig. 5) using the global keys and receives k_0^i for $i \in [N]$ and k_1^i for $i \in [2, N]$ from $\exp_1^{sec}(\kappa)$. $\mathcal{S}$ generates k^* honestly.
2. In the Proof phase, for given message msg:
   (a) $\mathcal{S}$ computes $str = \mathsf{RO}(\mathsf{mvnd}||\mathsf{msg}||k^*)$. Furthermore, $\mathcal{S}$ records all the randomness used by $\mathcal{A}$ from the set of queries made by $\mathcal{P}$ and $\mathcal{V}_i, i \in [2, N]$ to RO.
   (b) $\mathcal{S}$ samples $\boldsymbol{\rho} \in \{0,1\}^\ell$, generates $[\![\rho]\!]_q, [\![\boldsymbol{\rho}]\!]_2$ where $\rho = \sum_{i=1}^{\ell} 2^{i-1} \cdot \boldsymbol{\rho}[i]$. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{MV\text{-}edaBits}}$ by sending $[\![\rho]\!]_q, [\![\boldsymbol{\rho}]\!]_2$ to the correspond $\mathcal{P}$ and $\mathcal{V}_j$ for $j \in [2, N]$.
   (c) $\mathcal{S}$ evaluate the circuit $\tilde{\mathcal{C}}$ cooperated with other parties:
      – In the subroutine Assign, $\mathcal{S}$ records all the values (i.e., $\boldsymbol{\mu}[1], \ldots, \boldsymbol{\mu}[\tilde{t}] \in \mathbb{F}_2$) and their corresponding MAC tags (i.e., $\boldsymbol{m}_\mu \in \mathbb{F}_{2^\kappa}^{\tilde{t}}$) by emulating PCF for $\mathcal{A}$. Here, $\mathcal{S}$ can define the corresponding MAC keys of these values (i.e., $\boldsymbol{k}_\mu \in \mathbb{F}_{2^\kappa}^{\tilde{t}}$).
      – $\mathcal{S}$ executes Step 4.(a) as an honest verifier.
      – $\mathcal{S}$ receives $\{[\![\omega^*_{\alpha,j}]\!]_2, [\![\omega^*_{\beta,j}]\!]_2, [\![\omega^*_{\gamma,j}]\!]_2\}_{j\in[\tilde{t}]}$ from $\mathcal{A}$ on behalf of the functionality $\mathcal{F}_{\mathsf{MV\text{-}CheckMULs}}$, if $\forall j$, s.t. $\omega^*_{\alpha,j} \cdot \omega^*_{\beta,j} = \omega^*_{\gamma,j}$, and $\exists i$, s.t. $[\![\omega^*_{l,i}]\!]_2, \in \{\alpha, \beta, \gamma\}$ is not a valid IT-MAC, sends abort to $\mathcal{V}_i$ and sends true to all the other $\mathcal{V}_j, j \in [2, N]\backslash\{i\}$ on behalf of $\mathcal{F}_{\mathsf{MV\text{-}CheckMULs}}$.
   (d) Upon opening $[\![\boldsymbol{c}]\!]_2$, if any $\boldsymbol{c}[j]$' IT-MACs is invalid, $\mathcal{S}$ aborts. Otherwise, $\mathcal{S}$ continues.
   (e) $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

Indistinguishability of the simulation is obliviously based on the PCF assumption (Definited in Section 2.7): the only non-syntactic difference between the simulation and the real protocol is that when $c \neq \sum_{i=1}^{\ell} 2^{i-1} \cdot \boldsymbol{r}[i] + \sum_{i=1}^{\ell} 2^{i-1} \cdot \boldsymbol{\rho}[i]$ with $\boldsymbol{r} \neq \mathsf{PRF}_{\mathsf{dk}}(\mathsf{msg})$. As the $\mathsf{CheckMuls}$ subroutine guarantees that the $c = r + \rho$ is correctly computed from binary parts of edaBits and PRF evaluation except with the negligible probability $\mathsf{negl}(\kappa)$. On the other hand, $\mathcal{F}_{\mathsf{MV\text{-}edaBits}}$ guarantee red ρ (aggregated by binary representations of $\mathsf{mv\text{-}edaBits}$) is consistent with blue ρ (generated from integer part of $\mathsf{mv\text{-}edaBits}$). Therefore, we have $[\![R]\!]_q = [\![r]\!]_q \cdot G$ except negligible probability.

□

D Security Proof of Theorem 3

This section presents the security proof of stateless deterministic multi-party EdDSA signing protocol, as shown in Fig. 15. Similarly, this security proof begins by *correctness* where an EdDSA signature (R, σ) could be verifiably generated. Then, the *privacy* will be proven via UC-framework 2.2 on the basis of plain EdDSA (which has been proven in [BCJZ21,BDD23]). There is a slightly gap between plain EdDSA and our $\prod_{\mathsf{MP,Sign}}$ where $R = \mathsf{PRF}_{\mathsf{dk}}(\mathsf{msg}) \cdot G$ in the plain

EdDSA scheme and $R = \sum_{i\in[1,n]} R_i = \sum_{i\in[1,n]}(\mathsf{PRF}_{\mathsf{dk}_i}(\mathsf{msg}) \cdot G)$ in our protocol. However, as PRF is a pseudorandom function, the value R in the plain scheme and the values $R_1, \ldots, R_n$ in the secure protocol are pseudorandom, both under the constraint that is fixed with the same message. Thus, these R_is are computationally indistinguishable except with probability $\mathsf{negl}(\lambda)$.

Theorem 7. *(Restate of Theorem 3) Assume that* PRF *is a pseudorandom function. Then,* $\prod_{\mathsf{MP},\mathsf{Sign}}$ *UC-realizes* $\mathcal{F}_{\mathsf{EdDSA}}$ *in the presence of an adversary statically corrupting up to* $n-1$ *parties, in the* $(\mathcal{F}_{\mathsf{MV\text{-}ND}}, \mathcal{F}_{\mathsf{Com}}, \mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{DL}}})$*-hybrid random oracle model.*

Proof. We begin by showing that $\prod_{\mathsf{MP},\mathsf{Sign}}$ computes $\mathcal{F}_{\mathsf{EdDSA}}$ (all honest parties running the protocol generate the correct signature). This holds since when all parties are honest, we have:

$$R = \sum_{i\in[n]} R_i = \sum_{i\in[n]} r_i \cdot G = \sum_{i\in[n]} \mathsf{PRF}_{\mathsf{dk}_i}(\mathsf{msg}) \cdot G$$

$$\begin{aligned}\sigma = \sum_{i\in[n]} \sigma_i \mod q &= \sum_{i\in[n]} s_i \cdot \mathsf{H}(\mathsf{pk}, R, \mathsf{msg}) + r_i \mod q \\ &= (\sum_{i\in[n]} s_i) \cdot \mathsf{H}(\mathsf{pk}, R, \mathsf{msg}) + (\sum_{i\in[n]} r_i) \mod q\end{aligned}$$

Thus, (R, σ) would be a valid signature with $r = \sum_{i\in[n]} \mathsf{PRF}_{\mathsf{dk}_i}(\mathsf{msg})$ and $\mathsf{pk} = \sum_{i\in[n]} s_i \cdot G$.

We now proceed to prove security and consider the case that $n-1$ parties are corrupted. Similarly, let P_1 denote the honest party and $P_i, i \in \mathcal{I} = [2, n]$ denotes the set of corrupted parties. First, the simulator $\mathcal{S}$ needs to extract the corrupted party's input in order to send it to the trusted party. As we will show, this is possible by the fact that in the $(\mathcal{F}_{\mathsf{MV\text{-}ND}}, \mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{DL}}}, \mathcal{F}_{\mathsf{Com}})$-hybrid model $\mathcal{S}$ receives the secret keys s_i and dk_i from $\mathcal{A}$. We always implicitly assume that $\mathcal{S}$ passes all communication between adversary $\mathcal{A}$ and environment $\mathcal{Z}$. Simulating this protocol for an adversary corrupting P_i is done as follows:

Key Generation:

1. The simulator $\mathcal{S}$ extract sk_i from the set of queries made by P_i to H.
2. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{MV\text{-}ND}}$ for $\mathcal{A}$ by recording all the values $(\mathsf{mvzk\text{-}input}, i, \mathsf{dk}_i)$ that are received by $\mathcal{F}_{\mathsf{MV\text{-}ND}}$ from $\mathcal{A}$.
3. $\mathcal{S}$ also emulates the functionality $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{DL}}}$ by sending $(\mathsf{proof\text{-}receipt}, sid_{pk,1})$ to the adversary $\mathcal{A}$ and recording the values $(\mathsf{com\text{-}prove}, sid_{pk,i}, \mathsf{pk}_i, s_i)$ that are received by $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{DL}}}$ from $\mathcal{A}$.
4. Upon receiving pk from $\mathcal{F}_{\mathsf{EdDSA}}$, $\mathcal{S}$ computes $\mathsf{pk}_1 = \mathsf{pk} - \sum_{i\in[2,n]} \mathsf{pk}_i$ and sends $(\mathsf{decom\text{-}proof}, sid_{pk,1}, \mathsf{pk}_1)$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{DL}}}$.

5. $\mathcal{S}$ receives the messages $(\mathsf{decom\text{-}proof}, sid_{pk,i})$ that $\mathcal{A}$ sends to $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathrm{DL}}}$, if $\mathsf{pk}_i \neq s_i \cdot G$ in the associated $\mathsf{com\text{-}prove}$ values of Step 2 above, then $\mathcal{S}$ sends abort to $\mathcal{F}_{\mathsf{EdDSA}}$, outputs whatever $\mathcal{A}$ outputs and halts. Else, $\mathcal{S}$ stores all the values $\{\mathsf{dk}_i, \mathsf{sk}_i, s_i, \mathsf{pk}_i, \mathsf{pk}\}_{i\in[2,n]}$.

Signing: Upon receiving the message msg

1. If msg has previously been seen, $\mathcal{S}$ reuses the value R_1, σ_1 stored in the memory from the last time. Otherwise, $\mathcal{S}$ proceeds to the next step.
2. The simulator $\mathcal{S}$ receives $(\mathsf{nonce}, \mathsf{msg}, R)$ from $\mathcal{F}_{\mathsf{EdDSA}}$. $\mathcal{S}$ computes $R_1 = R - \sum_{i\in[2,n]} \mathsf{PRF}_{\mathsf{dk}_i}(\mathsf{msg}) \cdot G$.
3. Upon receiving $(\mathsf{mvzk\text{-}verify}, did_1, \mathsf{msg}))$ that $\mathcal{A}$ sends to $\mathcal{F}_{\mathsf{MV\text{-}ND}}$, $\mathcal{S}$ sends $(\mathsf{mvzk\text{-}proof}, \mathsf{msg}, [\![R_1]\!]_q)$ to $\mathcal{A}$.
4. $\mathcal{S}$ receives $(\mathsf{mvzk\text{-}prove}, did_i, \mathsf{msg})$ from $\mathcal{A}$, if $R_i \neq \mathsf{PRF}_{\mathsf{dk}_i}(\mathsf{msg}) \cdot G$, then sends $(\mathsf{mvzk\text{-}proof}, \mathsf{msg}, res)$ to $\mathcal{A}$ with $res = \perp$.
5. If no $res = \perp$ happens, $\mathcal{S}$ sends $(\mathsf{proceed}, \mathsf{msg})$ to $\mathcal{F}_{\mathsf{EdDSA}}$ and receives $(\mathsf{msg}, (\sigma, R))$ in response.
6. $\mathcal{S}$ simulates $(\mathsf{receipt}, sid, 1)$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{Com}}$ and receives $(\mathsf{commit}, sid, i, \sigma_i^*)$ from $\mathcal{A}$.
7. $\mathcal{S}$ computes the signature share $\sigma_1 = \sigma - \sum_{i\in[2,n]}(s_i \cdot \mathcal{H}(\mathsf{pk}, R, \mathsf{msg}) + r_i) \mod q$ with $r_i = \mathsf{PRF}_{\mathsf{dk}_i}(\mathsf{msg}) \mod q$ and sends $(\mathsf{decommit}, sid, 1, \sigma_1)$ to $\mathcal{A}$.
8. Upon receiving $(\mathsf{decommit}, sid, i)$ from $\mathcal{A}$ sent to $\mathcal{F}_{\mathsf{Com}}$, $\mathcal{S}$ instructs $\mathcal{F}_{\mathsf{EdDSA}}$ to send $(R, (\sum_{i\in[2,n]} \sigma_i^* + \sigma_1))$ to P_1. If $\sigma_i^* \neq s_i \cdot H(\mathsf{pk}, R, \mathsf{msg}) + r_i \mod q$, $\mathcal{S}$ aborts. Otherwise, it stores the records $(\mathsf{msg}, \sigma, R, \{\sigma_i, R_i\}_{i\in[2,n]}, \sigma_1, R_1)$ in the memory.

Indistinguishability of simulation. We show that the simulation by $\mathcal{S}$ in the ideal model results in a distribution identical to that of an execution of $\prod_{\mathsf{MP,Sign}}$ in the $(\mathcal{F}_{\mathsf{MV\text{-}ND}}, \mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathrm{DL}}}, \mathcal{F}_{\mathsf{Com}})$-hybrid random oracle model.

The simulation of the key generation phase is merely syntactically different from the real protocol. Note that $\mathcal{S}$ successfully extracts s_i from the query of $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathrm{DL}}}$. In the simulation of the signing phase, the actual values obtained by the corrupted party P_i during the execution are $\mathsf{pk}, \mathsf{pk}_1$(in the key generation), nonce R_1 and signature share σ_1. The distribution of these values in a real execution is

$$R_1 = \mathsf{PRF}_{\mathsf{dk}_1}(\mathsf{msg}) \cdot G, R = R_1 + \sum_{i\in[2,n]} R_i,$$
$$\sigma_1 = s_1 \cdot h + r_1 \in \mathbb{Z}_q, \sigma = \sigma_1 + \sum_{i\in[2,n]} \sigma_i \in \mathbb{Z}_q,$$

where dk_1 are random but fixed in the key generation phase, the same in all signing executions.

The distributions over these values in the simulated execution are

$$R_1 = R - \sum_{i\in[2,n]} \mathsf{PRF}_{\mathsf{dk}_i}(\mathsf{msg}) \cdot G, R,$$
$$\sigma_1 = \sigma - \sum_{i\in[2,n]} (s_i \cdot h + r_i) \mod q, \sigma,$$

where dk_i are fixed in the key generation phase, and $R = \mathsf{PRF}_{\mathsf{dk}}(\mathsf{msg}) \cdot G, \sigma = s \cdot h + r \mod q$ correctly computed by $\mathcal{F}_{\mathsf{EdDSA}}$. Observe that the simulation does not know dk and s, but this is the distribution since it is derived from the output from $\mathcal{F}_{\mathsf{EdDSA}}$.

As PRF is a pseudorandom function, the value R_1 in the real protocol and the values R_i, R in *both* protocols are pseudorandom, under the constraint that is fixed with the same message. In the simulation, $R_1 = R - \sum_{i\in[2,n]} R_i$ is also pseudorandomly and set with the same message. Thus, these R_1s are computationally indistinguishable except with probability $\mathsf{negl}(\lambda)$.

Finally, in the real protocol, we have the following holds

$$\sigma_1 \cdot G = h \cdot \mathsf{pk}_1 + R_1$$

Similarly, in the simulation, since $\mathsf{pk} = \mathsf{pk}_1 + \sum_{i\in[2,n]} \mathsf{pk}_i$, $R = R_1 + \sum_{i\in[2,n]} R_i$ and $\sigma = \sigma_1 + \sum_{i\in[2,n]}(s_i \cdot h + r_i) \mod q$, and σ, R are correct signature received from $\mathcal{F}_{\mathsf{EdDSA}}$, we have

$$\begin{aligned}\sigma_1 \cdot G &= \sigma \cdot G - \sum_{i\in[2,n]} (s_i \cdot h + r_i) \cdot G \\ &= h \cdot (\mathsf{pk} - \sum_{i\in[2,n]} \mathsf{pk}_i) + (R - \sum_{i\in[2,n]} r_i \cdot G) = h \cdot \mathsf{pk}_1 + R_1,\end{aligned}$$

Thus, these σ_js are identical.

If $\mathcal{S}$ does not abort and msg is first called, then we have $\sigma^* = \sigma_1 + \sum_{i\in[2,n]} \sigma_i$ where σ_i is received from $\mathcal{A}$, which has the same distribution as the output in the real protocol. That is if any modified $\sigma_i^* \neq s_i \cdot h + r_i \mod q$, the honest party will abort by checking $\sigma_i^* + \sum_{j\neq i} \sigma_j \neq h \cdot \mathsf{pk} + R$ except with probability $\mathsf{negl}(\lambda)$, just as what the $\mathcal{S}$ behaves in Step 8.

Thus, the simulator $\mathcal{S}$ aborts in the ideal-world execution only if the real-world execution aborts. Furthermore, this also implies that the outputs of honest parties have the same distribution in both executions. In conclusion, any unbounded environment $\mathcal{Z}$ cannot distinguish between the real execution and ideal execution, except with probability $\mathsf{negl}(\kappa)$. This completes the proof. □